CRANFIELD UNIVERSITY

Defence Academy - College of Management and Technology

Department of Informatics and Systems Engineering,
Centre for Autonomous Systems

PhD THESIS
Academic Year 2009 - 2012

Jiyoung Choi

Model Checking for Decision Making Behaviour of Heterogeneous Multi-Agent Autonomous System

Supervisor:
Professor Antonios Tsourdos

January 2012

This thesis is submitted in partial fulfillment of the requirements for the Doctor of Philosophy

Abstract

An autonomous system has been widely applied for various civil/military research because of its versatile capability of understanding high-level intent and direction of a surrounding environment and targets of interest. However, as autonomous systems can be out of control to cause serious loss, injury, or death in the worst case, the verification of their functionalities has got increasing attention. For that reason, this study is focused on the verification of a heterogeneous multi-agent autonomous system. The thesis first presents an overview of formal methods, especially focuses on model checking for autonomous systems verification. Then, six case studies are presented to verify the decision making behaviours of multi-agent system using two basic scenarios: surveillance and convoy. The initial system considered in the surveillance mission consists of a ground control system and a micro aerial vehicle. Their decision-making behaviours are represented by means of Kripke model and computational tree logic is used to specify the properties of this system. For automatic verification, MCMAS (Model Checker for Multi-Agent Systems) is adopted due to its novel capability to accommodate the multi-agent system. After that, the initial system is extended to include a substitute micro aerial vehicle. These initial case studies are then further extended based on SEAS DTC exemplar 2 dealing with behaviours of convoy protection. This case study includes now a ground control system, an unmanned aerial vehicle, and an unmanned ground vehicle. The MCMAS successfully verifies the targeting behaviours of the team-level unmanned systems. Reversely, these verification results help retrospectively improve the design of decision-making algorithms by considering additional agents and behaviours during four steps of scenario modification. Consequently, the last scenario deals with the system composed of a ground control system, two unmanned aerial vehicles, and four unmanned ground vehicles with fault-tolerant and communications relay capabilities. In conclusion, this study demonstrates the feasibility of model checking algorithms as a verification tool of a multi-agent system in an initial design stage. Moreover, this research can be an important first step of the certification of multi-agent autonomous systems for the domains of robotics, aerospace and aeronautics.

To

My beloved father, mother,

my handsome husband, and
my lovely daughter

Acknowledgements

First of all, I appreciate my supervisor, Prof Antonios Tsourdos, for his thoughtful consideration and valuable guidance all through this journey. I would like to thank Dr Peter Silson and Prof Brian White for reviewing my study and giving me precious comments about my research. I would also like to thank Dr Seunk-Keun Kim who provided valuable advice for revision of my papers and Dr Robert Alexander who offered me the related research in the inital stages of my study. Especially, I appreciate Graham Watson for giving me the motivation and ideas on my research scenario.

I would like to thank SEAS DTC for giving me an opportunity to do a PhD.

Contents

List of Tables

List of Figures

Nomenclature

All units are in SI unless otherwise stated

Alphanumeric

L	Labelling function
R	Accessibility relation
W	A set of possible worlds

Abbreviations

CTL	Computational Tree Logic
DTC	Defence Technology Centre
FSA	Finite State Automata
GCS	Groud Control System
ISPL	Interpreted Systems Programming Language
LTL	Linear Temporal Logic
MAV	Micro Aerial Vehicle
MCMAS	Model Checker for Multi-Agent Systems
MoD	Ministry of Defence
MSC	Message Sequence Chart
OBDD	Ordered Binary Decision Diagram
PLTL	Propositional Linear Temporal Logic
PN	Petri Net
RTTL	Real-Time Temporal Logic
SEAS	Systems Engineering for Autonomous Systems

SMV	Symbolic Model Verifier
SPIN	Simple Promela Interpreter
UAV	Unmanned Aeial Vehicle
UGV	Unmanned Ground Vehicle

Chapter 1

Introduction

1.1 Aim of research

> An autonomous system is capable of understanding higher level intent and direction. From this understanding and its perception of its environment, such a system is able to take appropriate action to bring about a desired state. It is capable of deciding a course of action, from a number of alternatives, without depending on human oversight and control, although these may still be present. Although the overall activity of an autonomous unmanned aircraft will be predictable, individual actions may not be.

This is a definition of autonomous systems [1]. To use such autonomous systems has attracted extensive attention to many civil/military applications as Unmanned Aerial Vehicles (UAVs). They are commonly used for the tasks associated with four elements: *dull*, *dirty*, *dangerous* and *deep* [2]. Surveillance mission, a role of communication relay, or acting as a air-to air refuelling tanker can be considered as *dull* tasks. *Dirty* tasks are missions in contaminated environments by CBRN (Chemical, Biological, Radiological, or Nuclear) elements. For instance, UAVs are being applied for the reconnaissance over civilian fire locations that is dangerous to human because of smoke and flames. Moreover, UAVs are recently committed to the battlefield, for example, Predator in Libya, for the performance of *dangerous* tasks. Lastly, *deep* tasks are kinds of penetrating enemy territory to complete observation and attack missions. As various tasks of autonomous systems get introduced, safety concerns to the operations in a general public area have been raised because autonomous systems can be out of control enough to cause loss, injury, or death to persons or property. Safety cannot be guaranteed simply by good design because the any behaviour of system may be upset by mistakes made during its production, installation or use[52]. Furthermore, to apply autonomous systems to more complex missions the structure of systems has got more complicated, which increases the possibility of malfunctions. Therefore,

aviation communities in most of the countries have focused on the regulatory authorities and certification to supervise the manufacture and operation of autonomous systems. Federal Aviation Administration in USA issued a *Notice of Policy on Unmanned Aircraft in the National Airspace System* in 2007 [3] and has focused on the development of certification standards to make unmanned aircraft system uniformly certified. SEAS DTC in UK has also been focusing on certification, especially for safety-related issues, in their research scope [4]. This study has been motivated by this demand of certification for autonomous systems. Since the research about the regulation for autonomous systems is in the initial phase, the most important part for the current certification issues is to abstract the essential properties which must be valid in autonomous systems in realistic application scenarios. For that reason, this thesis deals with several practical scenarios and aims to move forward towards the verification of heterogeneous autonomous systems.

1.2 Related Work

Beyond a single-agent autonomous system, a group of autonomous systems could be more reliable due to inherent redundancy to unexpectedness. Here, the agent means autonomous program which should select an proper action that is expected to maximise its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge it has[5]. In this background, a multi-agent autonomous system has been recently considered as a substitute to the single-agent autonomous system for higher mission successfulness[6]. In the reference [7], a team of UAVs were used to search an area of interest that contains regions of opportunity and potential hazard cooperatively. Kloetzer and Belta [8] developed a hierarchical framework for mission planning and control of robotic groups. Moreover, mission planning and optimization for multiple UAVs has been studied by using Linear Temporal Logic (LTL) [9, 10]. The temporal logic was used to perform motion planning for robots as well [11, 12]. For complicated systems, there have been attempts to verify the reliability of them at the design level by using a simulation in a virtual environment [13], a test with a mock-up, and formal methods. Firstly, the test with a mock-up costs a lot of money and time and does not guarantee the safety during the operation. Although the sim-

ulation costs less than the mock-up test, it is not easy to consider all the possible scenarios and situations. On the other hand, the formal methods are based on solid mathematical techniques and offer quantifiable answers to questions related with reliability of systems. Chaudemar *et al.* proposed a formal specification of the layered architecture of the UAV control system within the safety analysis using the Event-B method [14]. The Event-B is one of formal languages developed to specify and model the complex system with refinement mechanism [15]. In the reference [16], a formal approach to reactive system design was presented using POLIS [17]. Another formal approach, reverse engineering, was used to describe a case study involving a mission control system developed by the NASA Jet Propulsion Laboratory to command an unmanned spacecraft in [18]. Model-checking [19] is an automatic technique based on formal methods for verifying a finite state system to automatically check whether the target system satisfies the required properties or not. There are three sub-processes in model-checking: modelling, specification, and verification. The system to be verified is abstracted and translated into a language used in a model checker. The properties which we are interested in are specified by means of temporal logic. Finally, the model checker verifies the system properties and gives the result with yes or no on the system satisfying them. In [, baie08] model-checking technique is compared to a computer chess program. A model checker examines all possible systems scenarios in a systematic manner like a computer chess program checks all possible moves. In this way, model-checking provides exhaustive verification ability that makes far more wide-reaching states verified than testing and simulation. Thus, model checkers are widely used to verify the safety-critical or high-autonomy systems. SPIN [20, 21] and SMV [22] are the most widely-used model checkers and have pros and cons respectively. SPIN was originated from automated protocol validation techniques introduced by Holzmann to deal with industrial-size problems [23]. Holzmann and *et al.* studied about new reduction strategies for conventional reachability analysis, as used in [23], to reduce either run time or memory requirements by four to six times [24] and this study based the SPIN. Although the SPIN was developed to deal with protocol validation, the application area of it is not limited in computer science. Havelund *et al.* [25] used SPIN to verify the multi-threaded plan execution module of a spacecraft control system. The SPIN was also used for the verification of a complex and cooperative UAV monitoring task [26]. SMV was developed to avoid the state explosion problem

in computer science subject [27], however, its application has been very wide. In [28], a software system for Traffic Alert and Collision Avoidance System II (TCAS II) was analysed using the SMV. The SMV was also used to verify the discrete-state algorithms [29] and Computational Tree Logic (CTL) specifications for a Statechart [30]. Moreover, Pecheur *et al.* [31, 32] used the SMV for the development of autonomous controllers and the verification of diagnosability based on Livingstone, a model-based health-monitoring system, that can detect and diagnose anomalies and suggest possible recovery actions. While SPIN uses an automaton approach and only LTL formulae, SMV is a symbolic model checking tool and can mainly deal with CTL problems. Choi [33] investigates the explicit model checker SPIN on commercial flight guidance systems based on her prior work with the use of the symbolic model checker NuSMV [34]. There are the other model checkers for more complex system. Mohamed and Hans-Michael used SESA to check the validation of distributed multi-agent reconfigurable embedded control systems [35]. Furthermore, Du *et al.* [36] proposed a new class of labelled Petri net (LPN) for the testing and analysis of the obligations and accountability of participants in cooperative systems. Bordini *et al.* [37, 38] introduced AgentSpeak(F) to verify a multi-agent system and associated it with the SPIN and JPF [39]. In references [40, 41], Siminiceanu and Ciardo used SMART which is a software package providing a seamless environment for the logic and probabilistic analysis of complex systems. In addition to the temporal logic, modal logics including modal operators to reason about knowledge, beliefs, and strategies are considered for model checking. The tool MCMAS [42] is one of the recently-developed model checkers that can reason about time, knowledge, and correct behaviours of agents. Raimondi *et al.* [43] verified diagnosability and recoverability of Livingstone models using the MCMAS. Furthermore, Molnar and Veres [44] used the MCMAS for the verification of autonomous underwater vehicles as well. This research for verification of autonomous systems using model checkers can be a cornerstone of certification of autonomous systems. Webster *et al.* have showed the feasibility of using formal methods within the certification of UAVs for civil airspace [45, 13].

1.3 Contributions of the thesis

This thesis kindly shows the whole cycle of model checking for verifying multi-agent system and modifying system/property using verification result. The multi-agent systems dealt in this thesis are all not from book but from the real world problem. Therefore, the systems have more number of agents than those dealt in the previous research [46, 47, 48] and consist of heterogeneous agents. These four aspects can make this thesis be novel in the related research area.

The main contribution of this thesis is the demonstration of the feasibility of model checking for verifying the decision-making logic of multi-agent systems in the design-level. Although many researchers have mentioned about the feasibility of model checkers in the design-level, it is hard to find the studies that directly focus on that purpose. In this thesis, each scenario is inspired by the previous one. The first scenario related to a surveillance mission has been enhanced by means of adding the substitute MAV in the second scenario. This complementary measure comes out from the analysis of the verification result in the first scenario. The convoy scenario has been also modified in each phase to reflect the mistake found in the previous phase or to add supplementary agents/behaviours. As a result, the final scenario deals with seven agents, two environmental agents, fault-tolerant behaviours, improved safety properties, and communications relay behaviours. All these processes show how the users can apply a model checking algorithm to improve system design. Furthermore, by extending each scenario not only the lessons learned are transferred, but also some behaviours already verified before do not have to be analysed again if they are not related with a modification of system/property for the extension. Therefore it can lead to an efficient and practical design tool for verification of autonomous systems.

The second key contribution of this study is that the multi-agent systems dealt in this thesis are captured from the real world problem. The first multi-agent system in Chapter 3 is reproduced from MoD Grand Challenge. The Grand Challenge was a competition held by Ministry of Defence in UK. Because performance of this competition was assessed by not only innovative idea but also by implementation techniques, the multi-agent system used in the competition was practical and realistic. Also, the basic scenario for multi-agent system used in Chapter 4 to 7 is originated SEAS DTC Exemplar 2. Exemplar 2 is a scenario candidate to guide

researchers in their study related to military problem focusing on realistic issues. The multi-agent system in Exemplar 2 is intended to be of practical use and the events considered in Exemplar 2 is probable in real battlefield. By using these practical and reasonable multi-agent system, this thesis demonstrates that academic competence is successfully applied to solve real-world, industry-supplied problems.

Furthermore, this thesis represents the scalability of the MCMAS which can verify the extended multi-agent system consisting of nine agents including various environmental elements. The previous model checkers, e.g. SPIN and SMV, have the limitation of handling many number of agents [46, 48]. From that reason, they have dealt with the multi-agent systems composed of only two/three agents and have not considered an environmental element. Many researchers using the SPIN or SMV have acknowledged the problems of state-space explosion due to capability limitation of those model checkers. However, the MCMAS deals with nine agents (including *Environment Agent*) successfully in this thesis. In the final system in Chapter 7, the *possible worlds* and *accessibility relations* become more complex and interactive, so the reachable states of that scenario exceeds twenty millions. Moreover, the MCMAS still has the possibility that it can cope with more agents depending on the performance of computing resources. This scalability is an absolute strength of the MCMAS, and this thesis exhibits it clearly and for the first time with various mission scenarios of multi-agent systems.

As above-mentioned, the multi-agent systems described in this thesis are heterogeneous. The heterogeneous multi-agent system is more complicated because the interactions among the agents are more elaborate, and the priority of properties becomes more important for correct behaviours. Despite inevitable complexity, the heterogeneous multi-agent system can perform various missions in a versatile way not only in the battlefields but also in the civil applications. The multi-agent system considered in this study consists of ground vehicles, aerial vehicles, and manned agents. Additionally, the events which can be faced in real situations with uncertainty are taken into account for an individual agent. This heterogeneous system can make us approach more practically and deal with realistic scenario.

1.4 Structure of the thesis

The remainder of this paper has the following structure: First of all, the background knowledge to understand model checking is presented in Chapter 2. Chapter 3 describes the surveillance mission by a single MAV and a GCS. The scenario of this chapter is defined in Section 3.1. System modelling using Kripke approach is explained in Section 3.2, and property modelling using CTL is described in Section 3.3. Then, verification process and discussion are shown in Section 3.4 in detail. The scenario composed of the MAV and GCS is extended to a multi-agent system including two MAVs and a GCS in Section 3.5. Furthermore, the process of modelling and verification are unfolded from Section 3.6 to Section 3.8. Chapter 4 to Chapter 7 deal with a new mission scenario for convoying the UGVs using the UAVs and GCS. The scenario starts with a single UGV, an UAV, and a GCS in Section 4.1 and the following sections are composed of system modelling in Section 4.2, property modelling in Section 4.3, and verification and discussion in Section 4.4. In Chapter 5, the multi-agent system is extended to a larger multi-agent system possessing two UAVs, four UGVs, and a GCS as explained in Section 5.1. Moreover, Chapter 6 and Chapter 7 show more complicated multi-agent systems with fault-tolerant and communications relay behaviour, respectively. These three chapters have the same sections describing system modelling, property modelling, verification and discussion as Chapter 4. Conclusions and future works are discussed in Chapter 8.

Chapter 2

Model Checking-Background Knowledge

2.1 Formal methods

2.1.1 Formal methods and Systems Life-cycle

Formal methods can be regarded as different things to different people. The term is originated in formal logic, but now used in computer science area to refer to a wide range of mathematically based computing activities [49]. In this thesis, the most proper definition of formal methods [50] is that 'A formal method is a set of tools and notations (with a formal semantics) used to specify unambiguously the requirements of a computer system that supports the proof of properties of that specification and proofs of correctness of an eventual implementation with respect to that specification.' Therefore, formal methods are not so much *methods* according to this definition. In [51], Rushby identifies four levels of rigour in the implementation of formal methods:

- Level 0: No use of formal methods Specification documents are written in natural languages, pseudo-code or a programming language, augmented with diagrams and equations. Verification is performed manually by reviewing and inspecting. Validation is based on testing that is determined by the nature of the requirements, the specification and program structure.
- Level 1: Use of concepts and notation from discrete mathematics The some parts of requirements and specification documents written in natural language are replaced with notations and concepts derived from logic and discrete mathematics. This does not represent a full adoption of a formal approach.
- Level 2: Use of formalised specification languages with some mechanised support tools Formalised specification languages provide standardized no-

tation for discrete mathematics. Hence they can usually provide some automated methods of checking for certain classes of faults. Proofs are normally performed informally and referred to as rigorous proofs. However, several methods in this level provide explicit formal logics of deduction that would permit formal proof, even manually.

- Level 3: Use of fully formal specification languages with comprehensive support environments, including mechanised theorem proving or proof checking The fully formal specification languages employ a strictly defined logic and provide methods for the use of formal proofs. The formal proving methods permits the use of mechanised techniques such as proof checkers and theorem provers. In this level, the probability of detection faults increases within the various descriptions of the system. Moreover, mechanised techniques effectively remove the possibility of faulty reasoning. However, the fully formal specification languages costs a lot of effort and money in use and are generally very restrictive.

The use of formal methods is motivated by the expectation that mathematical analysis can contribute to the reliability and robustness of a system [52]. To develop a reliable system formal methods can be used at various phases through the life-cycle of a system. The systems life-cycle is a process of developing, or sometimes altering information of, systems [53]. It can be divided into five phases: Design, Implementation, Testing, Evaluation, Analysis. System design can be constructed from motivation, conception, or requirements. After that, according to the design concept, system is developed and sometimes integrated with the other system. A developed system is now operated and tested for evaluation. Analysis of the evaluation can result in phase-out and disposal of the system, maintenance, or modifying the system design [54]. In life-cycle, a system can be validated its performance whether it satisfies the requirements or not in testing and evaluation phases. It means if there is mistake by a designer in an initial design phase, it is probably found in the testing phase. Therefore, if the system does not fulfill the requirements of developers by this mistake, its development life-cycle resumes from the design phase again. In that case, huge loss of effort and cost can be anticipated.

Therefore, to prevent the propagation of mistake in a design phase, formal methods can be applied to verification of a system in this level. The significance

of applying formal methods to a design-level certification of critical and real-time systems has been clearly understood among systems developers, and there has been an increased proliferation to study formal methods, not only in theory but also in application for autonomous systems. Applying the formal methods entails the construction of a high-level description or a mathematical model of the system of interest. The model can then be subjected to a variety of analyses such as simulation, model checking and performance evaluation. As such processes can be automated and performed iteratively, the formal methods approach can be incorporated into system design at an early stage and offers a systematic approach during the design-level with a great degree of automation. In this context, a systematic approach represents for investigating events, gathering new knowledge, and then correcting or integrating it with previous knowledge of a system. In order to apply formal methods to a system design-level the tasks to be performed are logically partitioned into three parts:

Modelling The system design or operating scenario is converted to a formalism that is acceptable to automated verification tools, called model checkers. In some cases, abstraction may be used to suppress unimportant details of the design.

Specification The next step is to state the necessary properties that the system must satisfy. Again, it is necessary to use a formalism that is acceptable to the model checker. In this process a temporal logic is usually used for hardware and software systems.

Verification The model checker then verifies the validity of the specification against the proposed model iteratively. The most common mode of operation is for these tools to verify a system's state-space for validity of the specifications and to produce verification results. For negative results produced by the tool, the counter example has to be analyzed, and then the problem can be traced back either to the model or specification incorrectness. After that, suitable amendment steps can be taken.

The outcome of this exercise is that the design has been proof-checked exhaustively, and all possible modes of system operation.

2.1.2 Formal modelling

Formal modelling techniques have originally evolved around the study of reactive systems. A reactive system/program constantly maintains an interaction with its environment rather than to result in some final value. The reactive programs widely include concurrent and real-time programs, embedded and process control programs, and operating systems such as a plane or a nuclear reactor. Because some reactive systems are not planned to terminate, they cannot be specified by a relation between initial and final states, but must be identified by their ongoing behaviour [55]. Therefore, its specification must be done in terms of its ongoing behaviour that changes with time. This definition closely tallies with an informal description of a robot application, and many researchers have considered applying these techniques to studying robot applications. To date several formal modelling techniques such as Message Sequence Charts (MSCs) [56], Finite State Automata (FSA) [57], Petri nets (PNs) [58], Kripke models and temporal logics have been proposed to specify, analyse, understand and verify the correctness of reactive systems.

The MSCs is a language for the description of message flow using graphic and text. It is usually applied for communicating and concurrent processes which can be easily represented as a message flow. Because only the explicit message flows can be specified by the MSCs, other details of behaviours must be deduced from the specification. Therefore, incomplete message specification can be used for more practical scenarios such as communication failure among multi-agent systems. However, the conventional MSC represents only a deterministic behaviour intrinsically. Deterministic property in this context means that for everything that happens there are conditions such that, given them, nothing else could happen [59]. Therefore, to conduct non-deterministic system high-level message sequence charts [60], which is extended from MSCs by allowing interaction and non-deterministic choices, are needed.

A model set used in the FSA, an automaton, consists of states, an initial state, an input alphabet, and a transition function that maps input symbols and current states to a next state. Although the FSA has an expressive power, in case multi-agent systems are required, explicit marking of states with channels and clocks, so called timed-automata, are essential in order to synchronize the whole systems.

Timed-automata [59] is equipped with a finite set of real-valued clock variables to model the behaviour of time-critical systems. However, the addition of channels and clocks makes automated systems complicated, large in size and deterministic. To express non-deterministic behaviour in addition, nondeterministic finite automaton (NFA) [61] is needed further. NFA is a canonical finite automata which is able to jump into several possible next states from each state and a given input alphabet.

Petri nets are an alternative method for modelling concurrent systems and computation community. The PNs are based on both mathematical and visual approach. A net is a bipartite directed graph consisting of *places* and *net-transitions* [55]. *Places* represent conditionals and *transition* represent events when a net models a system. Inherently, the Petri nets has a merit of modelling generality, but conventional Petri nets are not expressive enough to represent real-time system, and additionally including uncertainty in time. To tackle these type of systems, various type of canonical Petri nets are considered such as timed-Petri nets [62, 63, 64] and stochastic Petri nets [65].

The Kripke model is pioneer in the sense that every modal logics associates with not just a single interpretation, but a set of possible interpretations, called as *possible world* by S. Kripke in the 1950s and 1960s. This formalism is widely used in highly complex and zero fault tolerance problems such as verification of real-time software and correctness of logic-system design. The main advantage of this approach is its ability to represent real world uncertainty using a formal, yet intuitive model in the form of a directed graph.

2.1.3 Kripke modelling

Mathematically, a Kripke model [19] is represented a triple $M = (W, W_0, R, L)$ as shown in Table 2.1.

W	is the set of *possible worlds*
W_0	is the set of *initial possible worlds*
R	is a relation on W, ($R \subseteq W \times W$), *accessibility relation*
L	is a function $L : W \rightarrow P(atoms)$, *labelling function*

Table 2.1: The mathematical representation of Kripke model

Graphically, a Kripke model can be viewed as a directed graph, i.e., a set of labelled nodes, connected by directed edges as illustrated in Figure 2.1.

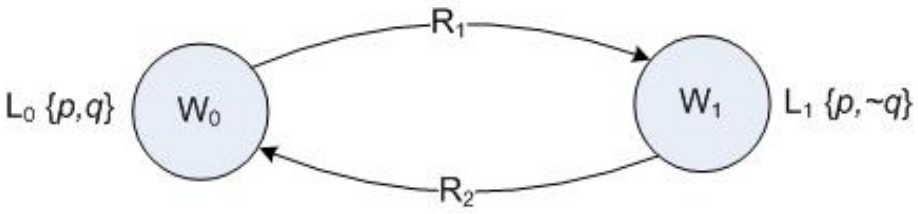

Figure 2.1: The graphical representation of Kripke model

The nodes are represented by *possible worlds*, the *labelling function* L_i denotes what holds in each of these nodes, and the edge R_i, represents *accessibility relation* for the transition between the *possible worlds*. Although these models are simple, they are expressive enough to capture temporal behaviours that are most important for reasoning about reactive systems. An expected behaviour of a system is tied down using the following mechanisms:

Possible worlds - The states that system components can stay at during operation.

Labelling functions - Propositional formulae representing unambiguous behaviours in each world.

Transition functions - Discrete conditions governing the traversals between worlds.

This kind of Kripke model is suitable for unambiguous modelling and representation of hybrid control approaches where discrete decision making such as obstacle avoidance, speed increase, flight-path change co-exist with continuous system dynamics. Another critical aspect of real-time systems is the order of task execution, also known as 'computation sequence'. The Kripke models allow us to capture this in a compact manner, unlike the FSA. Moreover, the same Kripke model can be reused to capture varying levels of granularity that is desired for validation. It is important to define the meaning of the term 'granularity of transitions' here. When a system switches from one mode of operation to another, the transitions ought to be such that they are atomic in the sense that no observable state of the system can result from executing part of a transition. If the transitions are too coarse, one may not include some states that are observable. Conversely, if the transitions are too fine, a state space explosion may occur or other spurious

errors may creep into the system. A state space explosion refers to the exponential rise in the number of states in the model related to the number of components that actually make up the state. With computational and algorithmic advances, modern software for model checking can handle a large number of states, so the issue is not the computational difficulty, but the challenge of retaining modelling transparency. Consequently, the main reasons for the success of the Kripke approach is its ability to represent real world uncertainty formally so it has been applied for complicated and safety-critical problems like real-time software [66, 67].

Additionally Kripke models can capture the temporal behaviour. However, because the representation of real system can be too complex and the size of states and variables become very large to describe the real system concretely, it is important to balance between the simple representation of model and the accurate reflection of the real system.

2.2 Formal specification: verifiable logic

2.2.1 Propositional logic

Propositional logic [59] is a two-valued logic, where formulae are assigned true or false values. Therefore, it can be regarded as a verifiable logic since the formulae can be checked whether they are true or not in some systems. The semantics of this logic is represented by using truth tables or inductive rules on the structure of the formula itself. Generally speaking, there are two types of truth value assertion: static and dynamic. If there is a fixed and time-independent truth value, the assertion is termed as static. In contrary, if an assertion is time-dependent, it is considered as dynamic. The propositional logic is also one of the logics that studies joining/modifying entire propositions, statements or sentences to form more complicated ones and deriving the logical relationships and properties by combining or altering the statements. Logical connectives such as the words 'and' and 'or', and the rules determining the truth-values of the propositions are involved in propositional logic. It also connects with the way of modifying statements, for example, using the addition of the word 'not' changes an affirmative statement into a negative statement.

2.2.2 Modal logic

A modal logic [68, 69] attempts to deal with modalities. 'Modal' means to qualify the truth of a judgment. Traditionally, there are three modes represented by the modal logic: possibility, probability, and necessity. The logics for dealing with a number of the related terms such as *eventually, formerly, can, could, might, may,* and *must*, are also called as modal logics. The representative branches of study in the modal logic are as follows: alethic modalities (necessity, possibility, and impossibility), epistemic logic (certainty), temporal logic, deontic logic, doxastic logic (belief), and so on.

2.2.3 Temporal logic

Temporal logic can be used to describe how the behaviours of systems unfold over time. In [70], Pnueli suggested a unified approach to program verification for sequential and parallel systems. The main proof methods suggested in this study was that of temporal reasoning about systems. And then, Manna and Pnueli firstly proposed the use of temporal logic for reasoning about the properties of reactive systems [55]. This logic uses a set of atomic properties, Boolean connectives, and four temporal operators. The temporal operators normally mean *future* operators: *invariant, eventually, next*, and *until*. There are two different underlying structures of time. The first one is Linear Temporal Logic (LTL) [59], and the second one is Computational Tree Logic (CTL) [71, 72]. The underlying structure of time in the LTL is a totally ordered set. On the other hand, in the CTL the underlying structure of time is branching tree-like as depicted in Figure 2.2.

The LTL uses four temporal operators: X (next), G (invariant), F (eventually), and U (until), and the CTL uses the temporal operators above mentioned with two path quantifiers: A (for all paths) and E (a path exists). Even though the CTL uses the path quantifiers additionally, the expressive power of the CTL and LTL are not comparable because there are the properties that can be expressed in the LTL but cannot be expressed in the CTL, and vice versa. But there is a temporal logic, CTL*, which is definitely more powerful than the CTL and LTL. It uses the temporal operators of the LTL and the path quantifiers of the CTL, but differently from the CTL its temporal operators can be used without path quantifiers. Therefore, CTL can be viewed as a fragment of CTL*. There is another fragment of CTL*,

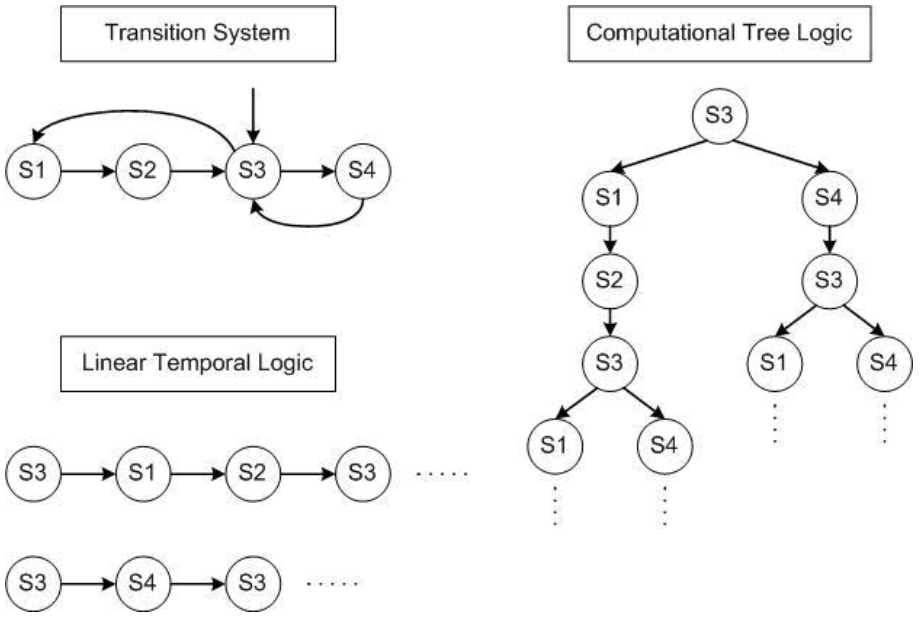

Figure 2.2: Representations of LTL and CTL for a given transition system

Propositional Linear Temporal Logic (PLTL) [57]. PLTL [73] is a linear time logic with past modalities Y (yesterday) and S (since). Because it does not use the path quantifiers like LTL, it only deals with the set of executions and can not examine alternative executions which split off from one state at each time step where a nondeterministic choice is possible. Based on this original temporal logic, Real-Time Temporal Logic (RTTL) is extended to apply automated verification to real-time distributed systems for safety-critical applications [74, 75].

2.3 Model checker

2.3.1 Why model checking?

Model checking[59, 57] can explore all possible systems states in a brute-force manner. In other words, model checking tool examines all possible system scenarios in a systematic manner and shows that a given system satisfies a certain property. There are some advantages of model checking which induce industry an increasing interest. Firstly, recently-developed model checking tools can handle state spaces of about 10^8 to 10^9 states with explicit state-space enumeration and larger state spaces up to $10^4$76 states for specific problems using clever algorithms

and tailored data structures. Moreover, model checking can reveal even the subtle errors that may remain undiscovered by testing and simulation because they are aimed at tracing the most probable defects. Additionally, the properties of interest can be checked individually when model checking is used. Therefore, users can focus on the essential properties at every phase of development. The autonomous multi-agent systems dealt with in this thesis can be considered as prototype. They are not fully-developed system and have potential to be extended. Hence, a verification tool for these systems must have ability to deal with increasing state space. Moreover, it needs to discover any error that can be created from evolution of system/specification and does not require complete specification in the initial phase of study. Consequently, model checking is the most proper technique to verify the systems in this study.

2.3.2 Definition and requirement of model checker

Model checking is an automated procedure that performs an exhaustive or symbolic search of the system's state-space and determines the truth value of the specification in question, i.e., an automated formal methods. Moreover, the model checking always terminates with a *'true'* or *'false'* answer and provides a counterexample which can be used to check the system errors and to take corrective actions. The algorithms used in model checking rely on the explicit construction of the finite state model targeted for verification. When applied to finite state systems, model checking can be performed automatically. In the case of complex systems, such an approach rapidly leads to the state explosion problem. Therefore, the principles underlying these model checking algorithms have evolved continuously, based on the type of temporal logic used in the specification. Usually, model checking involves satisfying the *never* claims, where the aim is to show that a (undesirable) specification (or condition) is never reached.

The model checking problem of a system involves the verification of certain fundamental system properties: *reachability*, *safety*, *liveness* and *fairness* properties. The details of each property follows as:.

- *reachability* property means that some particular property can be reached.

- *safety* property represents that, under certain conditions, something never occurs.
- *liveness* property means that, under certain conditions, something will ultimately occur.
- *fairness* property implies that, under certain conditions, something will (or will not) occur infinitely often.

Reachability is simple to express such as 'the program can enter a critical section' or 'the program cannot reach the crash state.' They can also be conditional like 'critical section can be entered without n = 0'. *Reachability* may also apply to any reachable state. LTL is poorly suited for specifying *reachability* properties because it implicitly quantifies over all computation paths. Thus, LTL can only express *reachability* negatively such as 'something is not reachable'.

Safety conditions are generally of the form 'both processes will not be in the critical section simultaneously.' In general, *safety* statements express that an undesirable event will not occur.

Examples of *liveness* include 'any request will ultimately be satisfied' or 'the program will terminate.' As the terminology suggests, the general meaning of the *liveness* property is to state that some event will occur in the end. From an utilitarian point of view, the *liveness* property yields no information.

A classic *fairness* statement is the 'if access to a critical section is infinitely often requested, then access will be granted infinitely often.' In practice, the *fairness* properties are very often used to describe some form of non-deterministic sequences, in particular when dealing with concurrent system [76]. Fairness assumptions are requisite to establish liveness properties or other properties implying that the system makes progress [59]. There are two different types of fairness constraints: strong fairness and weak fairness. Strong fairness represents that an activity infinitely often occurs, but is not necessarily always. In other words there may be finite periods during which the activity does not occur. On the other hands, weak fairness means that an activity continuously happens and no period is allowed in which the activity does not happen.

2.3.3 Model checking tools

Many researchers have used model checking to deal with finite state concurrent systems, especially to verify hardware and software such as complex sequential circuit designs and communication protocols [19]. Also, many automatic tools for model checking, i.e., model checkers have been developed for a long time. As examples of earlier model checkers, there are Simple Promela Interpreter (SPIN), Symbolic Model Verifier (SMV/NuSMV), UPPAAL, KRONOS, HYTECH and so on.

The SPIN is a model checker for the LTL and was developed for the verification of protocols and software in the 1980s. As its programming language is PROcess Meta Language (PROMELA), the user must translate the system of interest into the PROMELA. The SPIN is known as a mature tool but cannot handle unbounded data.

The SMV is one of the model checking tools using Ordered Binary Decision Diagrams (OBDD) and based on the model checking techniques for the CTL. NuSMV, a re-implementation of SMV, applies the bounded model checking methods for the LTL in addition to the OBDD for the CTL. In the same way as the SPIN, the SMV and NuSMV need to translate the system into their own input languages. Generally, these translations in the aforementioned tools require simplification or abstraction of the systems, by which false counter-examples can be caused.

UPPAAL[77] is an integrated tool for modelling, simulating, and verifying real-time systems. It is able to analyze networks of timed automata with binary synchronization. The timed systems are described using *graphical editor* part and then users can perform simulation of the systems and check the behaviour of the designed systems using *graphical simulator*. At last, *verifier* check the reachability properties of the systems.

KRONOS[78] is a model checker for the TCTL (Timed Computation Tree Logic)properties of a timed automaton. The timed automaton can be given in textual form starting from a system consisting of some components. After that, KRONOS computes the automaton corresponding to the synchronized product.

HYTECH[79] is developed to analyze linear hybrid automata at Cornell University. A set of linear hybrid automata synchronized by some common transitions can be used in HYTECH. It computes the subsets of the global state space

from the automata in a textual form. Subsequently, the result of the computation provides information about the behaviour of the system.

There are more recently evolved model checking tools. For instance, BLAST [80] was developed to check the temporal safety properties of C programs automatically. It verifies a program by the following steps. Firstly, a program is represented by using a set of control flow automata. And then BLAST constructs an abstract reachability tree to present a portion of the reachable state space of the program. Finally, it shows whether the error configuration is never reached or not. BLAST was developed by the University of California and can be used to prove the memory safety of C programs.

SLAM [81] verifies the temporal safety properties of Window APIs implemented in the C programs. The analysis engine of the SLAM is SDV (Static Driver Verifier) which checks whether a device driver correctly interacts with the Window operating system kernel or not. Its constructs a Boolean program which has the same control-flow structure of the original C program but consists of only Boolean variables. And then these Boolean variables track important predicates over the states in the original program using the predicate abstraction technique to perform reachability analysis and counter-example driven refinement.

Zing [82] is a framework for software model checking. This project aimed to build a flexible and scalable model checking infrastructure for concurrent software. It includes a modelling language, a state explorer, and an automatic model generator from common programming languages such as Visual Basic, C/C++, C#, and so on.

Java Path Finder [83] was started from the translation from Java to Promela. Nowadays it is based on a Java Virtual Machine, which is a state software model checker for Java. The same as other software model checker, it explores all execution paths of a program and find violations of properties.

MAGIC [84] is another model checker that can be applied to C programs. Its main task is to check the consistency between software specifications and implementations.

There is one more model checker named as MCMAS[42]. The goal of the MCMAS is the development of formal verification techniques for multi-agent systems using model checking techniques. The systems used in the MCMAS are deon-

tic interpreted systems. For the purpose of specification of multi-agent systems, $CTLKD - A^{D,C}$ formulae are used. The $CTLKD - A^{D,C}$ stands for Alternating-time Temporal Epistemic Logic (ATEL) extended with the operators for correct behaviour. It means that this multi-modal logic includes temporal, epistemic, correct behavioural and strategic operators.

2.3.4 Selection of model checking tool: MCMAS

In this thesis, the MCMAS is adopted to verify a heterogeneous multi-agent autonomous system because it is more suitable for multi-agent system than the other tools and has no limitation of dealing with a large multi-agent system composed of many autonomous agents. MCMAS [42] is based on OBDD and uses its own language ISPL to describe deontic interpreted systems. OBDD is a canonical graph of Boolean function and acquired by imposing an ordering on the Boolean variables and reducing the graph of the Boolean function [85]. Originally, binary decision diagrams were introduced by Lee [86], and further studied and made known by Akers [87] and Boute [88]. Based on their studies, the full potential for efficient algorithms derived from the data structure was introduced by Randal Bryant. He suggested a fixed variable ordering for canonical representation and shared sub-graphs for compression as above-mentioned[85, 89, 90].

An interpreted system which can be used in MCMAS is represented with a tuple IS as follows.

$$IS = \left\langle (L_i, Act_i, P_i, t_i)_{i\in\{1,\cdots,A\}}, (L_E, Act_E, P_E, t_E), I, V \right\rangle \tag{2.1}$$

where A is the number of agents in the system and E represents *Environment Agent* which can be used to describe the environment in which agents interact. L_i are local states of agent i, Act_i represents a set of actions, and P_i and t_i mean protocols $P_i : L_i \rightarrow 2^{Act_i}$ and evolution functions $t_i : L_i \times L_E \times Act \rightarrow L_i$ each, where $Act = Act_1 \times \cdots \times Act_A \times Act_E$. Additionally, I is a set of possible initial global states and V is an evaluation relation, $V \subseteq S \times AP$, in a given set of atomic propositions AP. As described in the equation of evolution functions, evolution of one agent is affected by not only its own local state and action but also local state and action of *Environment Agent* and actions of the other agents. By this way, MCMAS can

practice interactions among individual agents and environment to construct a multi-agent system.

The ISPL has six essential parts to represent an above-described interpreted system: *Environment Agent*, *Agent*, *InitStates*, *Evaluation*, and *Formulae*. In the *Agent* parts including *Environment Agent*, a Kripke model can be constructed using state variables, actions, protocols, and evolution. *Possible worlds* can be defined as state variables, *labelling function* is described in protocol section using properties defined as actions, and rules in evolution represent *accessibility relation* among *possible worlds*. In the *InitStates* part, the initial states of all agents are defined. Finally, the atomic propositions and their valid states are each declared in the *Evaluation* part. Using these propositions, the properties which we want to verify are expressed in the *Formulae* part.

Chapter 3

Surveillance Mission by a Single MAV

3.1 Scenario definition

To start with a simple multi-agent system, a part of decision making model used in the MoD Grand Challenge[1] is considered in this chapter. The Grand Challenge was a competition to look for innovative ideas and the technology to solve the military problem faced in a hostile urban environment. The scenario was that the competitors were in command of a troop operating apart from them and the troop made a reconnaissance to find threats. The competitors had already been aware of potential danger points where the enemy may lie in ambush, but, there were snipers, danger, and improvised explosive devices all around. Therefore, they had to discover whether the hostile urban environment is safe or not finding and removing hidden threats using intelligence graphic information gathered from multiple joint and coalition sources such as satellite and manned/unmanned aerial vehicle feeds. Figure 3.1 shows the town where the final demonstration was set up [91].

Cranfield University participated in this project as a member of the Stellar team and used the Kripke models to describe autonomous systems: Ground Control System (GCS), Micro Aerial Vehicle (MAV), High-Level Unmanned Aerial Vehicle (HLUAV), and Unmanned Ground Vehicle (UGV) which implemented missions for the MoD Grand Challenge [47]. These integrated mixed vehicle systems demonstrated by the Stellar team found targets with complete coverage and no false alarms. Moreover, despite the bad weather, Stellar team detected system failure and used their operators effectively during the search, so they could create a clear and accurate report upon completion of the task. As a consequence, Stellar team became the winner and won the RJ Mitchell Trophy as shown in Figure 3.2.

As an initial step, only the GCS and MAV are selected from the aforementioned Grand Challenge scenario to abstract the decision making model from one of their missions. An overview of the mission between the GCS and MAV is depicted in

[1] http://www.challenge.mod.uk

Figure 3.1: Sky view of Copehill Down

Figure 3.2: The Stellar team in Grand Challenge

Figure 3.3. The main mission of the MAV is to obtain images of potential threats. For this, the MAV follows the waypoints commanded from the GCS, captures the images and sends them back to the GCS. Then, the images obtained from the MAV can be used for threat analysis, which decides whether each threat needs to be verified further by the UGV. Moreover, the MAV will automatically land by following safety requirement in cases: 1) the communication channel to the GCS is lost, 2) the GCS sends a landing command.

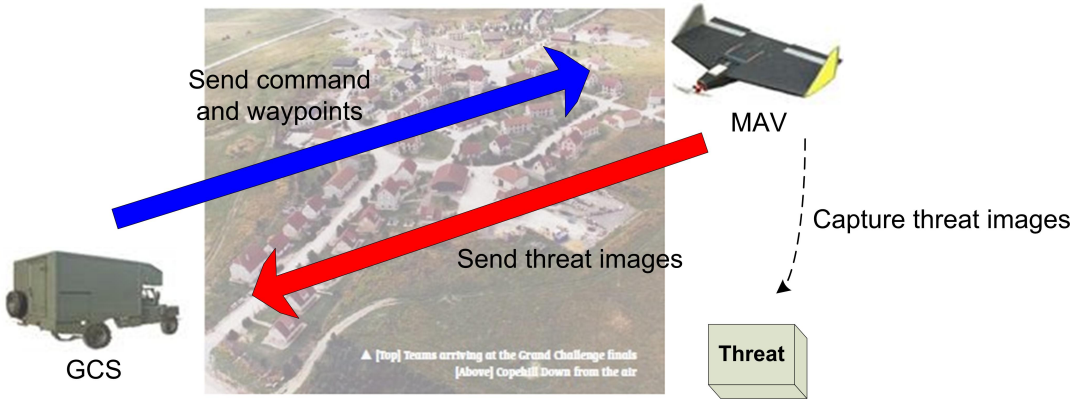

Figure 3.3: The scenario overview of surveillance mission by a single MAV in perspective of Grand Challenge mission

3.2 Kripke modelling of multi-agent system

3.2.1 Kripke model for MAV

The behaviours of the MAV can be defined as follows: launch, path-following, capturing images, sending images, and land. However, in a real situation, it is impossible for the MAV to transit from path-following to capturing/sending images state because the MAV follows the waypoints capturing or sending images simultaneously. Therefore, the behaviours performed in the same time need to be integrated to construct a correct Kripke model. Figure 3.4 shows the modified Kripke model for the MAV. In the state of the path-following the MAV performs a waypoint following manoeuvre, captures the target images and sends them to the GCS.

The *accessibility relations* are described as follows:

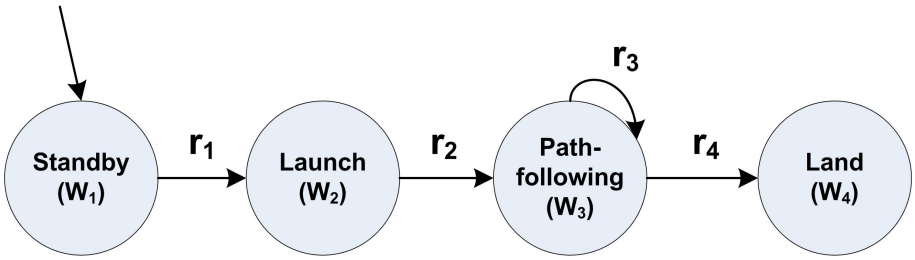

Figure 3.4: Kripke model for the MAV

- $(W_1 - W_2)$: The transition happens if the MAV receives the launching command.
$$r_1 = \begin{cases} true & \text{if the MAV receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if the MAV has launched successfully.
$$r_2 = \begin{cases} true & \text{if the MAV has been launched successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_3)$: The MAV follows the waypoints, captures and sends the target images to the GCS in normal situation.
$$r_3 = \begin{cases} true & \text{if the MAV performs path-following} \\ & \text{and its mission successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_4)$: The transition happens if the MAV needs to land.
$$r_4 = \begin{cases} true & \text{if the GCS sends the landing command} \\ & \text{or communication is lost.} \\ false & \text{otherwise} \end{cases}$$

3.2.2 Kripke model for GCS

To construct the Kripke model for the GCS, the behaviours of the GCS related with MAV mission can be defined as the following states: launch-commanding, waypoint-sending, threat analysis, path-planning, and land-commanding. In the path planning state, the GCS calculates the next waypoint that the MAV has to visit.

The transitions among the states can be described in detail:

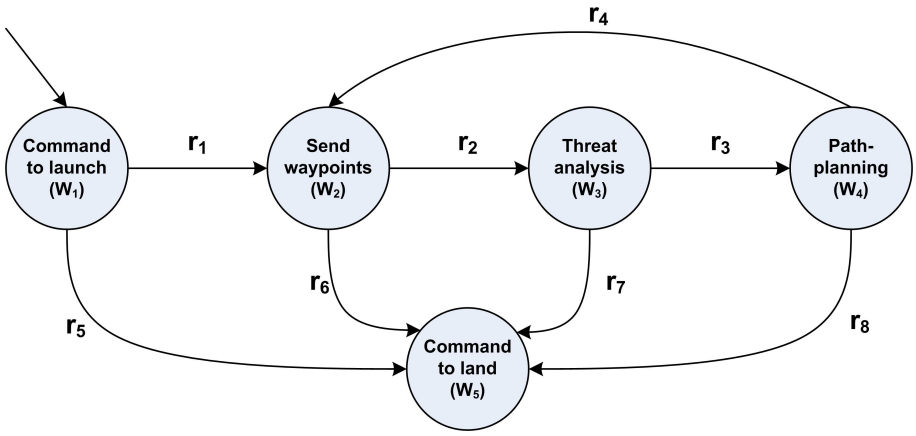

Figure 3.5: Kripke model for the GCS

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.
$$r_1 = \begin{cases} true & \text{if the GCS sends the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if the MAV sends the threat images.
$$r_2 = \begin{cases} true & \text{if the GCS receives the threat images from the MAV.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_4)$: The transition happens if the GCS has finished threat analysis.
$$r_3 = \begin{cases} true & \text{if the GCS has finished threat analysis.} \\ false & \text{otherwise} \end{cases}$$
- $(W_4 - W_2)$: The transition happens if the GCS has finished path-planning.
$$r_4 = \begin{cases} true & \text{if the GCS has finished the calculation of new waypoints.} \\ false & \text{otherwise} \end{cases}$$
- $(W_1 - W_5)$,$(W_2 - W_5)$,$(W_3 - W_5)$,$(W_4 - W_5)$: The transition happens if the GCS has decided to command landing to the MAV.
$$r_5, r_6, r_7, r_8 = \begin{cases} true & \text{if the GCS sends the landing command to the MAV} \\ & \text{because of the emergency or mission completion.} \\ false & \text{otherwise} \end{cases}$$

3.3 Modelling of properties to be verified

There are two properties to be verified in this scenario. The first property is *'The MAV must land or stay at standby state if the GCS sends the landing command or the*

communication channel is lost.' This is an important property in unmanned systems by the safety requirement. The second property to be verified is *'The MAV must be launched if the GCS sends the launching command successfully.'* This property is defined to check whether the MAV follows the command from the GCS correctly or not. These properties can be expressed in CTL as follows.

$$AG(((GCS.state = LC) \vee (Environment.state = LO)) \rightarrow AX((MAV.state = SB) \vee (MAV.state = LA))) \tag{3.1}$$

$$AG(((GCS.state = CL) \wedge (Environment.state = CO)) \rightarrow AX((MAV.state = L))) \tag{3.2}$$

where *'LC'* and *'CL'* mean *Command to Land* and *Command to Launch* states of the GCS, respectively. Similarly *'SB'*, *'L'*, and *'LA'* represent *Standby*, *Launched*, and *Land* states of the MAV. The states related with the state of communication are represented by *'LO'*, *Communication Lost*, and *'CO'*, *Communicating*.

3.4 Model checking with MCMAS

3.4.1 MCMAS model

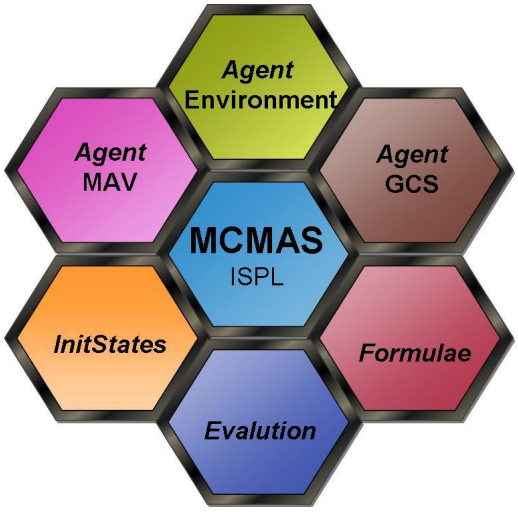

Figure 3.6: The elements of MCMAS model in Section 3.4.1

To perform the model checking automatically, the decision making behaviours of multi-agent system described in Section 3.2 are translated into ISPL, the input language of MCMAS. Figure 3.6 depicts the elements of the ISPL for the verification.

Figure 3.7 shows the entire ISPL code which represents all *Agents*, initial states of each agents, *Evaluation*, and *Formulae*. *Environment Agent* is constructed to represent the states of communication between the GCS and MAV. It has two states, communicating and lost, and available actions, send/receive and none. The Kripke models of the GCS and MAV are represented in *Agent* part by means of interpreted system formulation explained in Section 2.3.4. The *Agent GCS* consists of five states as described in the Kripke model: launch-commanding state, waypoint-sending state, threat analysis state, path-planning state, and land-commanding state. Each state has an action equivalent to their state and, additionally, commanding landing action for emergency. The states of the MAV in *Agent MAV* are standby state, launch state, path-following state, and land state. The standby state, launch state, and land state have only one action respectively which means state completion. While the path following state has three different actions: waypoints following, sending images to the GCS, and landing. As explained in Section 2.3.4, **Evolution** parts of *Agent GCS* and *Agent MAV*show that all evolution of individual agent are established under interactions of locals and actions possessed by every agents and environment at that phase. To help to find the interaction part in **Evolution**, local states and actions of environment and the other agent used for each agent's evolution are marked with yellow line in Figure 3.7. The properties discussed in Section 3.3 are described in *Formulae* as CTL expression and the atomic propositions used to describe these properties are defined in *Evaluation*. The initial states of the GCS and MAV are declared as launch-commanding state and standby state respectively as depicted in each Kripke model. In case of *Agent Environment*, communicating is assigned as the initial state for the nominal state of environment.

```
Agent Environment
    Vars:
       state : {CO, LO};
    end Vars
    Actions = {SR, none};
    Protocol:
       state = CO: {SR, none};
       state = LO: {none};
    end Protocol
    Evolution:
       state = CO if ( Action = SR );
       state = LO if ( Action = none );
    end Evolution
end Agent
Agent GCS
    Vars:
       state : {CL, SW, TA, PP, LC};
    end Vars
    Actions = {CL, SW, TA, PP, LC};
    Protocol:
       state = CL: {CL, LC};
       state = SW: {SW, LC};
       state = TA: {TA, LC};
       state = PP: {PP, LC};
       state = LC: {LC};
    end Protocol
    Evolution:
       state = SW if ( (Action = CL) or (Action = PP) );
       state = TA if ( (state = SW) and (MAV.Action = SI)
                                          and (Environment.Action = SR));
       state = PP if (Action = TA);
       state = LC if (Action = LC);
    end Evolution
end Agent
Agent MAV
    Vars:
       state : {SB, L, PF, LA};
    end Vars
    Actions = {SB, L, WF, CI, SI, LA};
    Protocol:
       state = SB: {SB};
       state = L: {L};
       state = PF: {WF, CI, SI};
       state = LA: {LA};
    end Protocol
    Evolution:
       state = SB if ( ( (state = SB) and (GCS.Action =LC) ) or
                          ( (state = SB) and (Environment.Action = none) ) );
       state = L if ( (state = SB) and (GCS.Action = CL) and
                                          (Environment.Action = SR) );
       state = PF if ( ( (state = L) and (GCS.Action = SW) and
                                    (Environment.Action = SR) ) or ( (state = PF)
                       and (Environment.Action = SR) and !(GCS.Action = LC)));
       state = LA if ( ( (GCS.Action = LC) and (Environment.Action = SR) )
                          or ( (Environment.Action = none) and (state = L) ) or
                                ( (Environment.Action = none) and (state = PF) ));
    end Evolution
end Agent
Evaluation
       homing if ( (GCS.state = LC) or (Environment.state = LO));
       landing if ( (MAV.state = SB) or (MAV.state = LA) );
       clauncching if ((GCS.state = CL) and (Environment.state = CO));
       launching if ( MAV.state = L );
end Evaluation
InitStates
       (GCS.state = CL) and (MAV.state = SB) and (Environment.state = CO);
end InitStates
Formulae
       AG(homing -> AX(landing) );
       AG(clauncching -> AX(launching));
end Formulae
```

Figure 3.7: ISPL code used for this chapter

3.4.2 Verification results

The properties described in Section 3.3 are verified by means of MCMAS model checking command, and the results of them are shown in Figure 3.8

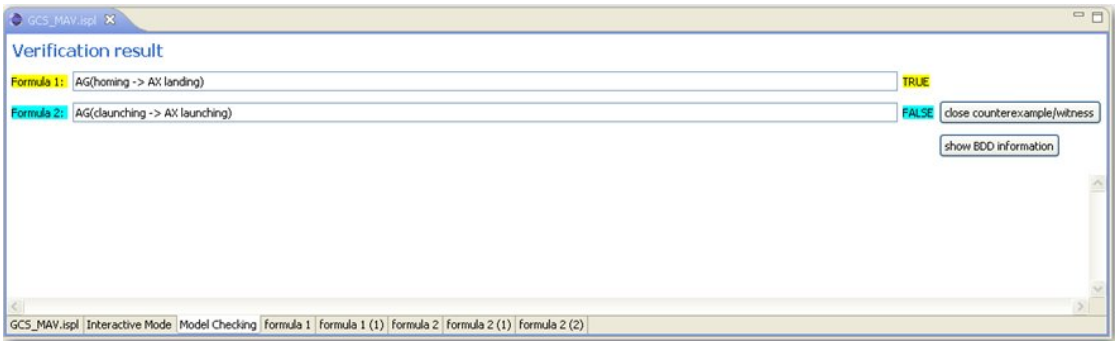

Figure 3.8: Verification results

Equation 3.1 has been verified as '*true*', but Equation 3.2 has '*false*' result. Using the counterexample/witness option, an error trace can be acquired and discussed in the next section.

3.4.3 Analysis and discussion

Figure 3.9: The counterexample of formula 2

Formula 2 is defined to check whether the MAV follows the command from the GCS correctly or not. Therefore, the MAV must be launched if the GCS sends the launching command successfully by designer's intention. However, the

verification result of formula 2 represents that this property is violated in some branch of the computational tree. The counterexample shows the case in which the communication has been lost after the GCS sent the launching command. In this situation, the MAV can not receive the command and can not help staying at the standby state. That is, the MAV can not launch because the GCS sent the launching command successfully but this command has not been sent to the MAV. This case has not yet been considered the stage of property modelling. In this scenario, communication between the agents is the most important part to perform cooperation correctly. Through the communication, agents share the information of targets, and transmit the command and waypoints. Therefore it is essential not only for mission execution but also for safety. However, the counterexample illustrated in Figure 3.9 showed that the mission cannot even be started when the communication has been lost. In real world, there is no almighty solution against communication loss. If communication between remote agents from each other is lost, the only option which they can take is to return to a base safely. Therefore, an interruption of mission execution by communication loss can not be conquered. Another possible case causing the interruption of mission completion is irrecoverable malfunction of agents. If agents have problems such as mechanical fault, disorder of electrical devices, or false of main controller and these problems can not be repaired for themselves, missions can not be executed any longer, either. Accordingly, the only alternatives for improving the probability of mission success are to use an additional agent or redundant communication channels.

3.5 Extended scenario definition

As a result of discussion in Section 3.4.3, the scenario is extended as follows: If the MAV malfunctions in the path following state, the GCS commands the MAV to land and a new MAV to launch. Then the new MAV performs the mission as a substitute for the old MAV. The extended model cannot solve the problem of communication loss basically, however, it can certainly improve the probability of mission completion. To construct the modified scenario the Kripke models of the GCS and MAV have to be changed with additional states.

3.6 Kripke modelling of extended scenario

3.6.1 Kripke model for MAV and substitute MAV

The new behaviour of the MAV aims to overcome malfunction during performing the mission. Figure 3.10 shows the modified Kripke model for the MAV and the substitute MAV.

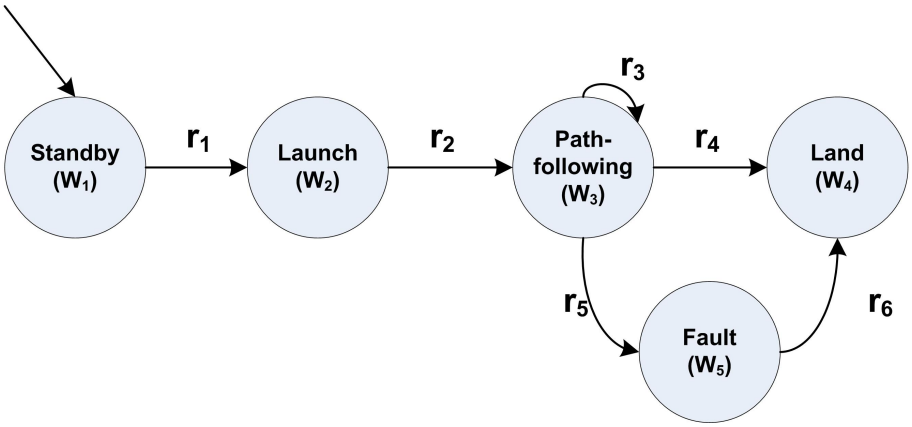

Figure 3.10: Kripke model for the MAV and the substitute MAV

The additional *accessibility relations* are described below:

- $(W_3 - W_5)$: The transition happens if the MAV detects its malfunction during path-following.
  $r_5 = \begin{cases} true & \text{if the MAV detects its malfunction during the mission.} \\ false & \text{otherwise} \end{cases}$
- $(W_5 - W_4)$: The transition happens if the GCS commands to land because of malfunction.
  $r_6 = \begin{cases} true & \text{if the GCS commands to land because of malfunction.} \\ false & \text{otherwise} \end{cases}$

3.6.2 Kripke model for GCS with additional behaviour

To combine the replacement behaviour into the original Kripke model for the GCS, the replacement state is added as depicted in Figure 3.11.

The additional *accessibility relations* are as follows:

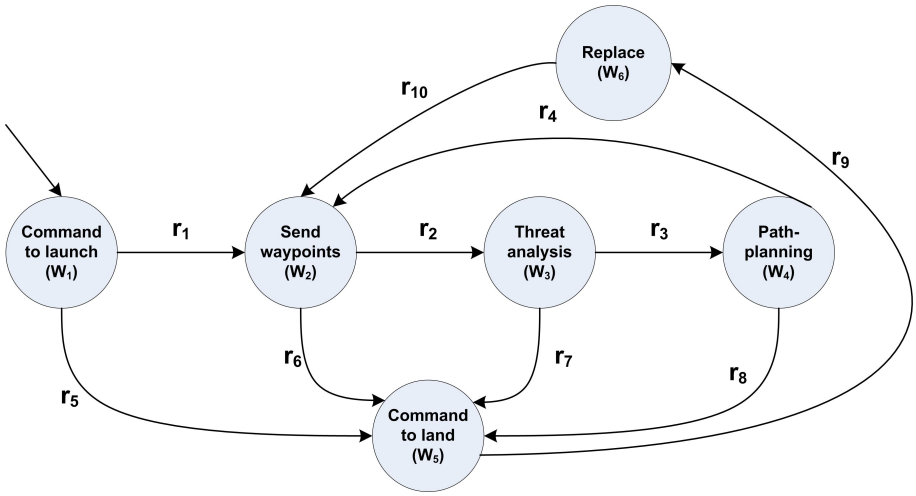

Figure 3.11: Kripke model for the GCS with replacement behaviour

- $(W_5 - W_6)$: The transition happens if the MAV malfunctions during the path-following mission.
  $r_9 = \begin{cases} true & \text{if the MAV malfunctions during the mission.} \\ false & \text{otherwise} \end{cases}$
- $(W_6 - W_2)$: The transition happens if the GCS commands the replacement of the MAV.
  $r_{10} = \begin{cases} true & \text{if the GCS commands the substitute MAV to launch.} \\ false & \text{otherwise} \end{cases}$

3.7 Property modelling of extended scenario

Three properties are specified to verify whether the replacement behaviour on the MAV is performed correctly or not. They can be expressed in CTL and interpreted as follows.

$$AX((MAV.state = MAL) \rightarrow (GCS.state = REP)) \tag{3.3}$$

which means *'If the MAV malfunctions, the GCS replaces it by the substitute MAV'* for the replacement mission.

$$AG(!((MAV.state = PF) \text{ and } (subMAV.state = PF))) \tag{3.4}$$

which means *'The MAV and the substitute MAV cannot fly simultaneously'* for the safety reason. Since this scenario assumes the path-planning is possible to be

designed only for one vehicle, two vehicles cannot perform the mission together.

$$AX((MAV.state = MAL) \rightarrow (subMAV.state = PF)) \tag{3.5}$$

This formula represents *'The subMAV will be launched only after the MAV malfunctions.'*

3.8 Model checking of extended scenario

3.8.1 MCMAS model

To reflect the changed Kripke models, the ISPL code has been modified in *Agent, Evaluation, InitStates*, and *Formulae* parts. The replacement state is added in the *Agent GCS*, and additional action/protocol and evolution functions related with the replacement state are attached. Same as the GCS, the modified *Agent MAV* has the malfunction state, action and arisen evolution functions additionally. Moreover the *Agent subMAV* has the same states, actions, and protocols, but evolution function because it has to be launched in case the GCS decides to execute replacement.

3.8.2 Verification results

MCMAS verifies the properties mentioned in Section 3.7, and the results of them are shown in Figure 3.12. All the results are *'true'*, which mean that the modified scenario for the replacement of faulty MAV is constructed successfully. Firstly, the GCS successfully decides to perform replacement of the MAV when it malfunctions. The substitute MAV is also launched only after the MAV malfunctions. Moreover, the result of formula 2 certifies that two MAVs do not fly simultaneously. This means there is no probability of collision between two MAVs.

3.8.3 Analysis and discussion

The system described in Section 3.1 was extended to include one more agent to improve the probability of mission completion. Furthermore, the modified system

Figure 3.12: Verification result of modified scenario

was modelled as the Kripke model easily on the base of the previous model and checked to prove the correct behaviour. The results of Equation 3.3 and 3.5 mean that if the MAV informs its fault, the GCS re-plans the mission scenario and the substitute MAV takes over the mission of the previous MAV successfully. These results perfectly accord with the design intention. Hereby, the probability of mission success can increase. Moreover, by checking the property described in Equation 3.4, it is clearly proved that the MAV and the substitute MAV do not fly simultaneously. This result also means that the additional behaviour of the multi-agent system does not violate the safety requirement. As mentioned in Section 3.7, this scenario assumed to perform path-planning for only one vehicle. If two MAVs fly simultaneously, and what is worse, one of them is faulty, a collision of two MAVs probably happens because they use or used same path according to the path-planning by the GCS. Therefore, formula 2 is very important for safety and its verification result fully satisfies the requirement.

The extension of the scenario in this chapter shows the feasibility of model checking process in the design level of a multi-agent system. By this way, a designer can find an unanticipated situation, which can not be easily found in a simulation, without a real test. Moreover, the result of this chapter showed that the modified system can be re-modelled and verified with less effort. In the following chapters, this strength of model checking will be explored more in detail with taking realistic complex scenarios into account.

Chapter 4

Convoy Mission by Single UAV and Single UGV

4.1 Scenario definition

The scenario used in this and subsequent chapters is referred from Exemplar 2 of the scenario candidates provided by SEAS DTC (Systems Engineering for Autonomous Systems Defence Technology Centre), UK [92]. Figure 4.1 depicts an overview of the mission scenario. The UAV airborne convoys UGV moving on the ground whose mission commanded by the GCS is to transport the resources towards the destination, e.g. a friendly military base in the battlefield. The UAV follows the predefined waypoints, captures the images along the route the UGV moves on and sends them back to the GCS until the UGV arrives at the destination. The main roles of the GCS are to analyze the images captured by the UAV and to re-plan the mission if necessary. There are some events which could potentially deter the mission completion: encountering with a GPS-denying area, a damaged route including roads or bridges, an intermittent communication loss, and pop-up threats on the ground. Each event directly affects the state transition of the agents by the decision making algorithms. However, the transition due to communication loss is different from the others. As discussed in Section 3.4.3, the communication loss can result in the termination of the mission. However, in a real battlefield, there might be an intermittent communication loss due to electro-magnetic interference, loss of line-of-sight, or enemy's jamming. Since the mission is not seriously interrupted by the intermittent communication loss, therefore, a distinction between intermittent and permanent communication losses is required to maximize the mission completion. In this scenario, the concept of confirmation time is applied for this distinction. The confirmation time is a threshold of accumulated time. If some event has been detected continuously for the predefined confirmation time, then an observer can declare an occurrence of the event. Generally, this concept is widely used to discriminate correct fault detection from false alarm [93].

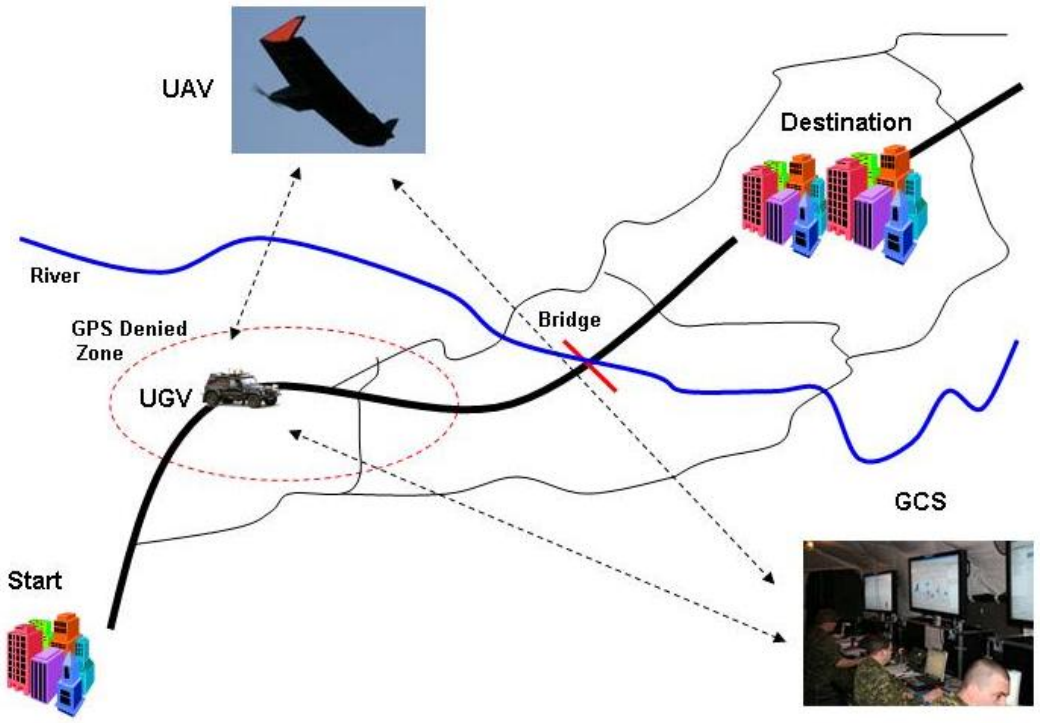

Figure 4.1: Convoy mission using single UAV and single UGV

4.2 Kripke modelling of multi-agent system

4.2.1 Kripke model for UAV

In the case of the UAV, the Kripke model is same as that of the MAV in Section 3.2.1. There are four *possible worlds*: standby, launch, path-following, and land. As mentioned before, because the behaviour of capturing and sending images is performed while the UAV follows the path, there is no state which represents capturing and sending the images additionally.

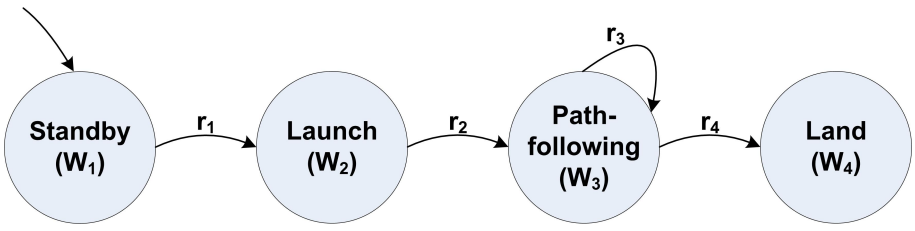

Figure 4.2: Kripke model for the UAV

The *possible worlds* and *accessibility relations* are as follows.

- $(W_1 - W_2)$: The transition happens if the UAV receives the launching command.
  $r_1 = \begin{cases} true & \text{if the UAV receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$
- $(W_2 - W_3)$: The transition happens if the UAV has launched successfully.
  $r_2 = \begin{cases} true & \text{if the UAV has launched successfully.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_3)$: The transition happens if the UAV performs the path-following.
  $r_3 = \begin{cases} true & \text{if the UAV performs the path-following.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_4)$: The transition happens if the UAV receives the landing command or the communication loss is confirmed.
  $r_4 = \begin{cases} true & \text{if the UAV receives the landing command} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$

4.2.2 Kripke model for UGV

There are seven *possible worlds* for the UGV with the following decision-making behaviours: ready, dispatched, path-following, obstacle avoidance, standby, go to the designated point, and arrive. The state of obstacle avoidance is considered due to pop-up threats on the route. Furthermore, differently from the UAV, the standby state is included because the UGV can stop moving when the GCS re-plans the mission. Similarly the state in which the UGV goes to the designated point is considered for the case of encountering with a GPS denying area and an intermittent communications loss. Figure 4.3 shows these *possible worlds* and their *accessibility relations*.

The *accessibility relations* are as follows in detail:

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.
  $r_1 = \begin{cases} true & \text{if the UGV receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$

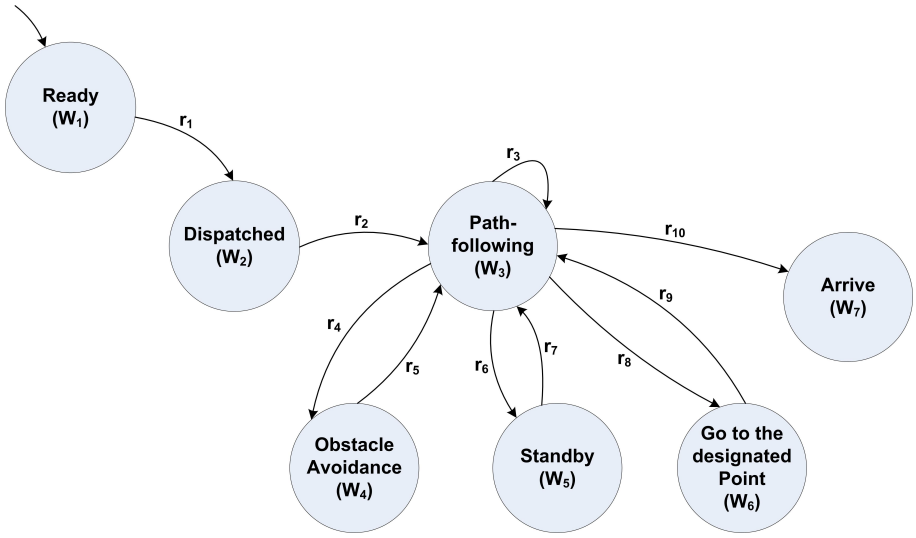

Figure 4.3: Kripke model for the UGV

- $(W_2 - W_3)$: The transition happens if the UGV has been dispatched successfully.

$$r_2 = \begin{cases} true & \text{if the UGV has been dispatched.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_3)$: The UGV keeps on the path-following state until it arrives at the destination unless there is a special event.

$$r_3 = \begin{cases} true & \text{if nothing happens and the mission keeps going on.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_4)$: The transition happens if the pop-up threats appear.

$$r_4 = \begin{cases} true & \text{if the pop-up threats appear.} \\ false & \text{otherwise} \end{cases}$$

- $(W_4 - W_3)$: The transition happens if the UGV has avoided the threats successfully.

$$r_5 = \begin{cases} true & \text{if the UGV has avoided the threats.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_5)$: The transition happens if the main route has been damaged or the communication loss has been confirmed.

$$r_6 = \begin{cases} true & \text{if the route has been damaged} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$$

- $(W_5 - W_3)$: The transition happens if the GCS re-plans the route.

$$r_7 = \begin{cases} true & \text{if the GCS re-plans the route.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_6)$: The transition happens if the UGV enters the GPS denial zone or an intermittent communication disorder happens.

$$r_8 = \begin{cases} true & \text{if the UGV enters the GPS denial zone} \\ & \text{or an intermittent communication disorder happens.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_3)$: The transition happens if the UGV gets out of the GPS denial zone and the communication has been restored.

$$r_9 = \begin{cases} true & \text{if the UGV gets out of the GPS denial zone} \\ & \text{and the communication has been restored.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_7)$: The mission has been completed if the UGV arrives at the destination.

$$r_{10} = \begin{cases} true & \text{if the UGV arrives at the destination} \\ false & \text{otherwise} \end{cases}$$

4.2.3 Kripke model for GCS

The *possible worlds* of the GCS related with decision-making behaviours are defined as four states: command to launch, image analysis, re-planning, and command to land. These *possible worlds* and their *accessibility relations* are represented graphically in Figure 4.4.

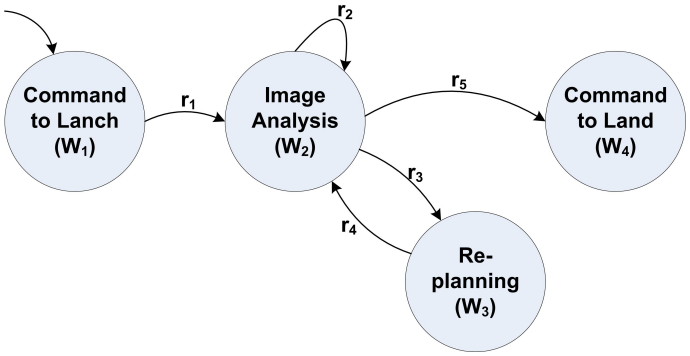

Figure 4.4: Kripke model for the GCS

The *accessibility relations* are as follows in detail:

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.
$$r_1 = \begin{cases} true & \text{if the GCS sends the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_2)$: The state maintains if the GCS performs the image analysis.
$$r_2 = \begin{cases} true & \text{if the GCS performs the image analysis.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if the GCS decides the re-planning.
$$r_3 = \begin{cases} true & \text{if the GCS decides the re-planning} \\ & \text{because of GPS denial or damaged route.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_2)$: The transition happens if the GCS has finished the re-planning.
$$r_4 = \begin{cases} true & \text{if the GCS has finished the re-planning.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_4)$: The transition happens if the mission has been completed or the communication loss is confirmed.
$$r_5 = \begin{cases} true & \text{if the mission has been completed} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$$

4.3 Modelling of properties to be verified

To check whether the decision-making behaviours among the agents are modelled properly, six properties related with decision-making behaviours are specified as follows.

$$AG((Environment.trigger = 1) \rightarrow AX(UAV.state = SB \vee UAV.state = Land)) \quad (4.1)$$

which means *'If the communication loss is confirmed, namely, the state of communication loss continues for three sequences, the UAV must land.'* In Section 4.2.1, the UAV executes launch when it has healthy communication state and to land when it loses communication permanently for safety reason. This formula is adopted to check whether this decision making algorithm of the UAV is executed accurately.

$$AG((Environment.trigger = 1) \rightarrow AX(!(UGV.state = PF))) \quad (4.2)$$

Same as the UAV, the UGV is planned to stop following the route when the communication loss is confirmed. The second formula is for verification of this behaviour.

$$AG((Environment.state = Comm\_Lost) \rightarrow AX(UAV.state = Land)) \tag{4.3}$$

which represents *'If the communication is lost, the UAV must land '* and contradicts the first property. Because the UAV is designed not to land by encountering the communication loss unless the communication loss has been confirmed. This property is defined to assure that there is no false decision making behaviour related with an intermittent communication disorder. The fourth property to be verified is that *'If the UGV comes into the GPS denial zone it must get out of the area and return to the previous point in which the GPS signal was valid'* and can be expressed in CTL as follow.

$$AG((Environment.state = GPS\_den) \wedge (UGV.state = PF) \rightarrow AX(UGV.state = GotoP)) \tag{4.4}$$

The fifth property is that *'If the UGV faces with the pop-up threats, then it performs obstacle avoidance maneouvre'*

$$AG((Environment.state = Popup) \wedge (UGV.state = PF) \rightarrow AX(UGV.state = ObsAvoid)) \tag{4.5}$$

The last property to be checked in this scenario is as follows.

$$\begin{aligned} &AG((Environment.state = Route\_dam) \\ &\rightarrow AX((GCS.state = Replan) \vee (GCS.state = Com\_land))) \end{aligned} \tag{4.6}$$

which states *'If the route is damaged the GCS must re-plan or end the mission.'* The latter three properties are all related with possible events in the environment and aimed to examine the correct behaviour of agents.

4.4 Model checking with MCMAS

4.4.1 MCMAS model

In this scenario, four agents are used to construct a multi-agent system in the battlefield. First of all, the events mentioned in Section 4.1 are described in

the *Environment Agent* part. Five variables representing the events are *None, Comm_lost, GPS_den, Route_dam,* and *Popup*. *None* means 'no event happens', *Comm_lost* means 'communication is lost', *GPS_den* represents 'the UGV is in a GPS denial zone', *Route_dam* represents 'the route is damaged', and *Popup* means 'the UGV encounters with pop-up threats.' *GPS_den*, *Route_dam*, and *Popup* directly affect the transition between states of the GCS and UGV as described in Section 4.2.3 and 4.2.2. The transition due to *Comm_lost* is different as explained in Section 4.1. Three temporary variables are used to save the past event for three sequences and a trigger variable is set up to represent that the communication loss is confirmed. The trigger variable turns to '1' from '0' when three temporary variables are all *Comm_lost*. The other agents consist the GCS, UAV, and UGV. Each has the states which represent *possible worlds* and the evolution rules which represent *accessibility relations* described in Section 4.2.

4.4.2 Verification result

The result of verification is shown in Figure 4.5. Differently from the other results, the result of formula 3 has to be *'false'* because the UAV must keep going on its behaviour against an intermittent communication loss until the communication loss is confirmed as explained in Section 4.3. Therefore, the result *'true'* of formula 1, 2, and *'false'* of formula 3 imply the decision making behaviours of the UAV related with communication loss are executed properly according to the confirmation logic. Formulae 4 and 5 acquire the result *'true'* as well, which shows the decision making behaviours of the UGV related with entering the GPS denial zone or encountering with the obstacle are performed correctly. From the last result, it is verified that the decision making behaviour of re-planning mission is successfully performed in the case of the event encountering with the damaged route.

4.4.3 Analysis and discussion

The scenario dealt with in this chapter can be regarded as the result of reflection from the previous chapter. In the previous chapter, MCMAS found the unexpected situation in which the MAV can not even start its mission due to com-

Figure 4.5: Verification Result

munication loss. Then, the multi-agent system was improved with the substitute MAV for mission completion. In this chapter, the other alternative is adopted for the mission completion. That is, the decision making algorithms linked with the communication loss are reinforced to accommodate more realistic scenarios by using the concept of confirmation time. The UAV and UGV used in this chapter do not take an action of mission termination such as *Land* against an intermittent communication loss until communication loss has been confirmed by confirmation time. These behaviours are successfully verified by MCMAS as described in Section 4.4.2. In addition to the reinforced communication problem, the scenario considers the other possible events in battlefield, which can deter the execution of the mission. To cope with these events, proper behaviours of the agents are designed such as re-planning and obstacle avoidance. The result of the formula 4-6 shows that these decision-making behaviours are also successfully constructed by design purpose. As a result, the convoy scenario can be estimated prosperously improving the mission success and well reflecting reality.

This enhanced system results in more interactive verifiable multi-agent system even though the number of agents in this chapter is three, equivalent to that of the previous chapter. Figure 4.6 depicts the Binary Decision Diagrams (BDD) information of the modified system in Section 3.5 and the system used in this chapter. The most valuable factor to be compared is the number of reachable states. As shown in the figure, the number of reachable states was increased from seventy five to three thousand, three hundred and eleven: approximately forty four times increase. However, the execution time is maintained as milliseconds. Moreover, the usages of memory are both around six/seven mega bytes. All the verification processes using MCMAS have been performed on Intel Core2 Duo 2.4GHz with 1.95GB of RAM running Windows XP. Taking this specification into consideration, the BDD information of two scenarios can be interpreted that

MCMAS has an ability to deal with a large sized system very efficiently with a reasonable computation requirement. This inference will be further investigated in the following chapters using bigger and more complex multi-agent system.

```
execution time = 0
number of reachable states = 75
BDD memory in use = 6440820
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 4004
**** CUDD non-modifiable parameters ****
Memory in use: 6440820
Peak number of nodes: 7154
Peak number of live nodes: 1411
Number of BDD variables: 30
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 10651
Number of cache hits: 5389
Number of cache insertions: 5486
Number of cache collisions: 126
Number of cache deletions: 0
Cache used slots = 2.04% (expected 2.07%)
Soft limit for cache size: 31744
Number of buckets in unique table: 7936
Used buckets in unique table: 51.46% (expected 51.12%)
Number of BDD and ADD nodes: 7114
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 6078
Number of dead ZDD nodes: 0
Total number of nodes allocated: 7114
Total number of nodes reclaimed: 9521
Garbage collections so far: 0
Time for garbage collection: 0.00 sec
Reorderings so far: 0
Time for reordering: 0.00 sec
```

(a) The result of Chapter 3

```
execution time = 0
number of reachable states = 3311
BDD memory in use = 7293492
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 9124
**** CUDD non-modifiable parameters ****
Memory in use: 7293492
Peak number of nodes: 56210
Peak number of live nodes: 9053
Number of BDD variables: 53
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 159361
Number of cache hits: 64897
Number of cache insertions: 94732
Number of cache collisions: 6314
Number of cache deletions: 66727
Cache used slots = 30.05% (expected 22.40%)
Soft limit for cache size: 97280
Number of buckets in unique table: 24320
Used buckets in unique table: 51.48% (expected 51.60%)
Number of BDD and ADD nodes: 19463
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 12405
Number of dead ZDD nodes: 0
Total number of nodes allocated: 94868
Total number of nodes reclaimed: 88754
Garbage collections so far: 4
Time for garbage collection: 0.00 sec
Reorderings so far: 3
Time for reordering: 0.14 sec
```

(b) The result of Chapter 4

Figure 4.6: Comparision of BDD information

Chapter 5

Convoy mission by a group of UAVs and UGVs

5.1 Scenario definition

The MCMAS manifested the feasibility for verification of multi-agent systems which have more than three agents through the result in Chapter 4. Accordingly, the system explored in this chapter will be expanded to include double agents and to take into account multiple-subagent behaviours such as taking over the duty between them. Fig. 5.1 depicts an overview of the enlarged system. The single airborne agent is changed to two UAVs which perform different missions, and the ground agent extends to have four members which behave with a leader-follower logic. UAV 1 flies along the main route capturing/sending the images like the single UAV used in the previous scenario. The UAV 2 convoys the UGV locally and can take over the mission of the UAV 1 when it malfunctions. The group of UGVs consist of head, middle 1, middle 2, and tail. The decision making behaviour of the head is totally the same as that of the UGV described in the previous chapter. However, the others change their states in accordance with the decision making of the leader. The events which were considered in the previous chapter are applied to this scenario as well including the distinction logic between intermittent and permanent communication losses.

5.2 Kripke modelling of multi-agent system

5.2.1 Kripke model for UAV 1

In the case of UAV 1, the Kipke model is similar to that of the UAV in Chapter 4, but the additional fault state need to be defined as shown in Figure 5.2. To establish the behaviour of taking over the mission within the group of UAVs, it

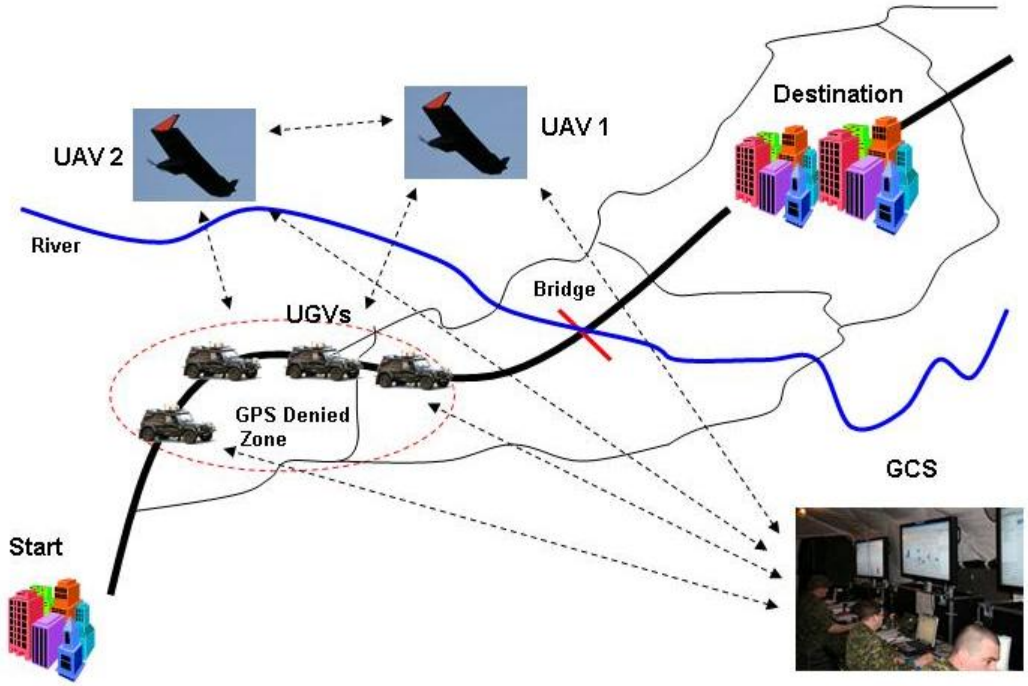

Figure 5.1: Extended system with a group of UAVs and UGVs

is assumed that UAV 1 has capability to detect its malfunctions for itself by using fault detection and diagnosis techniques.

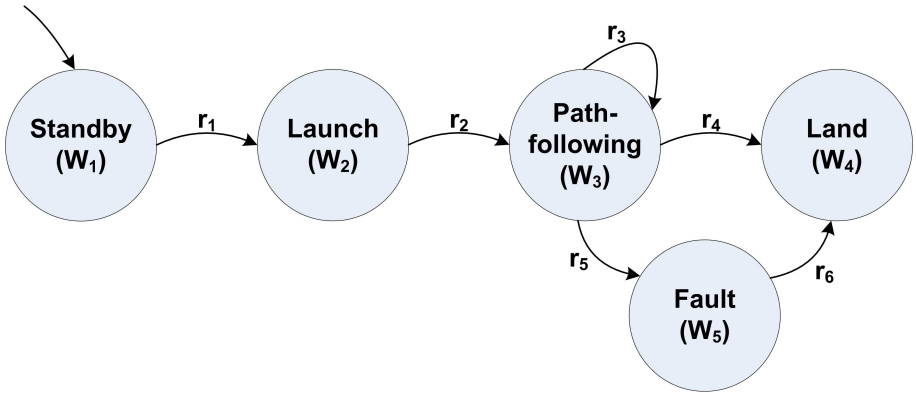

Figure 5.2: Kripke model for UAV 1

- $(W_1 - W_2)$: The transition happens if UAV 1 receives the launching command.

$$r_1 = \begin{cases} true & \text{if UAV 1 receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$

- $(W_2 - W_3)$: The transition happens if UAV 1 has launched successfully.
$$r_2 = \begin{cases} true & \text{if UAV 1 has launched successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_3)$: The transition happens if UAV 1 performs the path-following.
$$r_3 = \begin{cases} true & \text{if UAV 1 performs the path-following.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_4)$: The transition happens if UAV 1 receives the landing command or the communication loss is confirmed.
$$r_4 = \begin{cases} true & \text{if UAV 1 receives the landing command} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_5)$: The transition happens if UAV 1 self-detects its malfunctions.
$$r_5 = \begin{cases} true & \text{if UAV 1 detects its malfunctions.} \\ false & \text{otherwise} \end{cases}$$
- $(W_5 - W_4)$: UAV 1 lands if it is faulty.
$$r_6 = \begin{cases} true & \text{if UAV 1 is faulty.} \\ false & \text{otherwise} \end{cases}$$

5.2.2 Kripke model for UAV 2

UAV 2 has five *possible worlds* in the Kripke model: standby, launch, vehicle-following, land, and path-following as depicted in Figure 5.3. In this scenario, UAV 2 is designed to take over the mission of path monitoring when UAV 1 detects its malfunctions. For this, UAV 2 has the states of executing two different missions. The *accessibility relations* among *possible worlds* are represented as follows.

- $(W_1 - W_2)$: The transition happens if the head UGV is dispatched.
$$r_1 = \begin{cases} true & \text{if the head UGV is dispatched.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if UAV 2 has launched successfully.
$$r_2 = \begin{cases} true & \text{if UAV 2 has launched successfully.} \\ false & \text{otherwise} \end{cases}$$

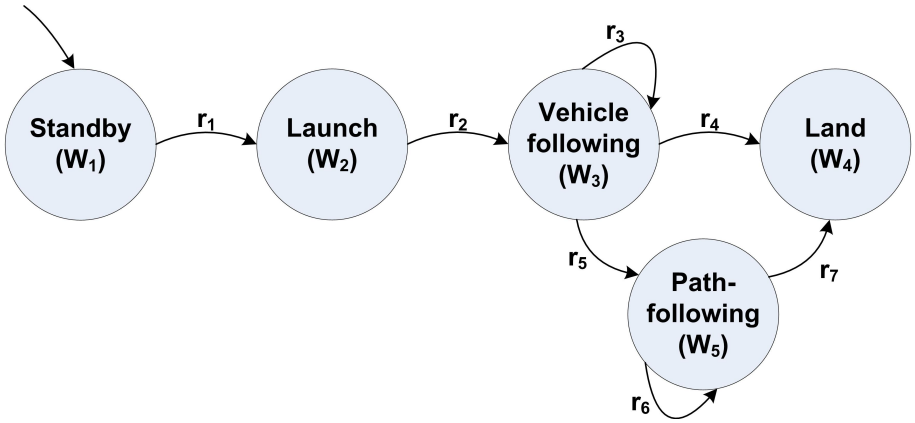

Figure 5.3: Kripke model for UAV 2

- $(W_3 - W_3)$: The transition happens if UAV 2 performs its mission without any problem.
  $r_3 = \begin{cases} true & \text{if UAV 2 performs the vehicle-following.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_4, W_5 - W_4)$: The transition happens if UAV 2 receives the landing command, the UGVs have arrived, or the communication loss is confirmed.
  $r_4, r_7 = \begin{cases} true & \text{if UAV 2 receives the landing command} \\ & \text{, the UGVs have arrived,} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_5)$: The transition happens if UAV 1 reports its malfunction.
  $r_5 = \begin{cases} true & \text{if UAV 1 reports its malfunction.} \\ false & \text{otherwise} \end{cases}$
- $(W_5 - W_5)$: The UAV 2 performs the path-following until it lands.
  $r_6 = \begin{cases} true & \text{if UAV 2 keeps performing the path-following.} \\ false & \text{otherwise} \end{cases}$

5.2.3 Kripke model for UGV

Figure 5.4 represents the Kripke model of the UGV. The Kripke model for the UGVs has the same *possible worlds* of the UGV modelled in Chapter 4 as follows: ready, dispatched, path-following, obstacle avoidance, standby, go to the designated point, and arrive. The *accessibility relations* of the head UGV and the previous

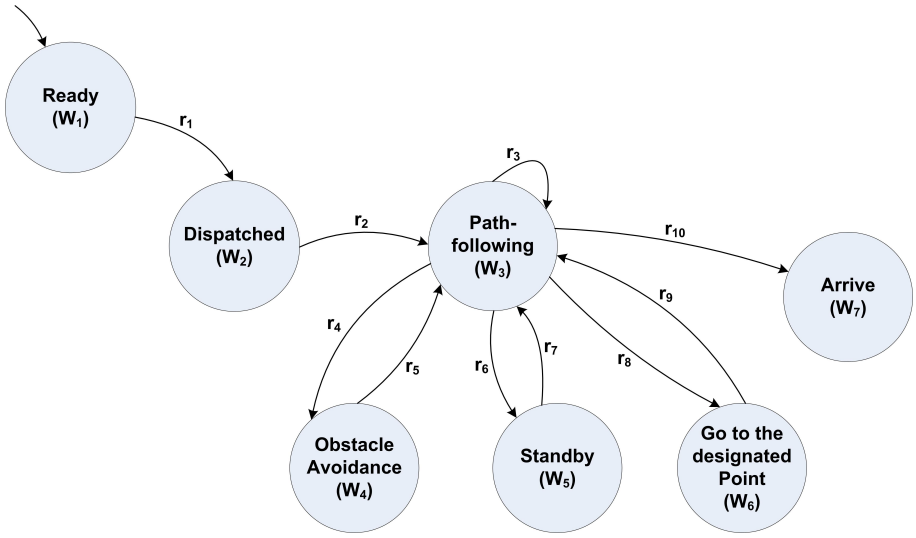

Figure 5.4: Kripke model for the UGV

UGV in Section 4.2.2 are also alike. However, because the other UGVs follow the transition of the leader, their *accessibility relations* have to be modified as follows.

- $(W_1 - W_2)$: The transition happens if the head UGV has been dispatched.
  $r_1 = \begin{cases} true & \text{if the head UGV has been dispatched successfully.} \\ false & \text{otherwise} \end{cases}$
- $(W_2 - W_3)$: The transition happens if the UGVs have been dispatched successfully.
  $r_2 = \begin{cases} true & \text{if the UGVs have been dispatched successfully.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_3)$: The UGVs keep on path-following until it arrives at the destination unless there is a special event.
  $r_3 = \begin{cases} true & \text{if nothing happens and the mission keeps going on.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_4)$: The transition happens if the pop-up threats appear.
  $r_4 = \begin{cases} true & \text{if the pop-up threats appear.} \\ false & \text{otherwise} \end{cases}$
- $(W_4 - W_3)$: The transition happens if the UGV has avoided the threats successfully.
  $r_5 = \begin{cases} true & \text{if the UGV has avoided the threats.} \\ false & \text{otherwise} \end{cases}$

- $(W_3 - W_5)$: The transition happens if the head UGV turns to the standby state.
$$r_6 = \begin{cases} true & \text{if the head UGV turns to the standby state.} \\ false & \text{otherwise} \end{cases}$$
- $(W_5 - W_3)$: The transition happens if the GCS re-plans the route.
$$r_7 = \begin{cases} true & \text{if the GCS re-plans the route.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_6)$: The transition happens if the head UGV goes to the designated point.
$$r_8 = \begin{cases} true & \text{if the head UGV goes to the designated point.} \\ false & \text{otherwise} \end{cases}$$
- $(W_6 - W_3)$: The transition happens if the head UGV get out of the GPS denial zone and the communication has been restored.
$$r_9 = \begin{cases} true & \text{if the head UGV get out of the GPS denial zone} \\ & \text{and the communication has been restored.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_7)$: The mission has been completed if the head UGV arrives at the destination.
$$r_{10} = \begin{cases} true & \text{if the head UGV arrives at the destination.} \\ false & \text{otherwise} \end{cases}$$

5.2.4 Kripke model for GCS

The Kripke model of the GCS does not need to be modified from the previous model in Section 4.2.3. Therefore, the details of *accessibility relations* are omitted.

5.3 Modelling of properties to be verified

To inspect if the extended scenario is constructed correctly and does not break the decision making algorithms which were verified formerly, the same properties in Section 4.3 are taken into account again. In addition to them, five more properties are appended to examine the correct behaviour among the group of UAVs/UGVs.

$$AG((UAVugv.state = Vfollowing \wedge UAVpath.state = Falut \\ \wedge\ !Environment.trigger = 1) \rightarrow AX(UAVugv.state = Pfollowing)) \tag{5.1}$$

$$AG(UGVhead.state = Standby \rightarrow AX(UGVtail.state = Standby)) \quad (5.2)$$

$$AG(!UGVtail.state = Arrive \rightarrow !UAVugv.state = Land) \quad (5.3)$$

$$AF(UGVtail.state = Arrive \rightarrow (UAVpath.state = Land \wedge UAVugv.state = Land) \quad (5.4)$$

$$AF(UGVhead.state = Arrive \rightarrow UGVtail.state = Arrive) \quad (5.5)$$

The first added property is that *'If UAV 1 malfunctions while UAV 2 is monitoring the UGVs with healthy communication, then UAV 2 must take over the mission of UAV 1.'* By checking this property, completion of the substitution mission in the air can be proved. In the group of the UGVs, the leader-follower logic must be investigated similarly to the UAV. For that reason, the property which means *'If the UGV head turns to the standby state, then the UGV tail must turn to the standby state.'* will be verified by the MCMAS as represented in Equation 5.2. The next property is that *'If the UGV tail has not arrived, then UAV 2 must not land.'* It is related with the safety property because the main role of UAV 2 is to monitor the group of the UGVs for their safety. Therefore, UAV 2 must land after the convoy of the UGVs arrives at the destination securely. In addition to the former property, the property related with mission termination will be verified as well. The most important mission in this scenario is to transfer the resources to the destination by the UGVs. Hence, if the UGVs completes the main mission, the group of UAVs must terminate their supporting mission. This property can be illustrated as *'If the UGV tail has arrived, then UAV 1 and 2 must land at last.' 'If the UGV head has arrived, the UGV tail must arrive at last.'* is the last property in this scenario. This property is modelled to investigate the functionality of the leader-follower logic among the UGVs as well.

5.4 Model checking with MCMAS

5.4.1 MCMAS model

Based on the MCMAS model in Section 4.4.1 the fault state of UAV 1, *Agent UAV 2*, and three UGVs are added for this new MCMAS model. Moreover, a *Fairness* section is supplemented from this scenario. This section contains Boolean expressions which must be *'true'* infinitely often along all executions. In this scenario, intuitively, it is required that the battlefield has no event infinitely often.

Therefore the proposition related with the state of *Agent Environment* is written in the *Fairness* section as follow.

$$Environment.event = None \tag{5.6}$$

5.4.2 Verification results

As shown in Figure 5.5 formula 1-6 have same result of the previous chapter. All the agents distinguished an intermittent communication loss from a permanent communication loss correctly and made proper decisions against with the special events. The result of formula 7 shows that the behaviour of taking over mission between UAV 1 and UAV 2 is executed successfully. Similar to this, the *'true'* results of formulas 8 and 11 stand for the correct execution of the decision-making algorithms among the UGVs by the leader-follower logic. As explained in Section 5.3, formula 10 aims to check the proper mission termination, and the *'true'* result proves that. The result which can not be accepted intuitively is that of formula 9. Using the counterexample option, the execution trace in which property 9 is broken is analyzed in the next section.

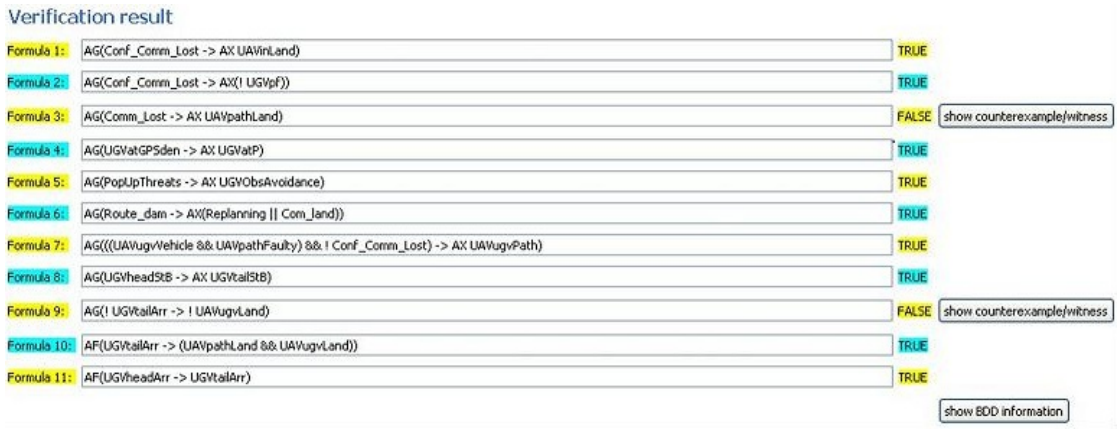

Figure 5.5: Verification Result

5.4.3 Analysis and discussion

The verification results of formulas 1-6 are not changed though the scenario has been extended. It demonstrates that the extension of the multi-agent system

does not frustrate the correct behaviours of the agents as expected in the design level. Because the addition of agents in the MCMAS model is almost like that of a plug and play device, an extension of a multi-agent system hardly violates basic properties. The *'true'* results of formulas 7, 8, and 11 assure the proper behaviours of the additional agents in groups of UAV and UGV, respectively. On the other hand, formula 10 represents the appropriate decision-making by a relation between the UAVs and the UGVs. The main mission of the UAVs is to convoy the group of UGVs unitl they arrive at the destination. Therefore, if the UGVs arrive at the destination, i.e., they achieve their goal, the mission of UAVs is completed as well. For the proper termination of the mission, the UAVs have to finish their flight when the tail UGV informs its arrival. Using MCMAS, all these behaviours are ascertained that they are performed correctly as designed.

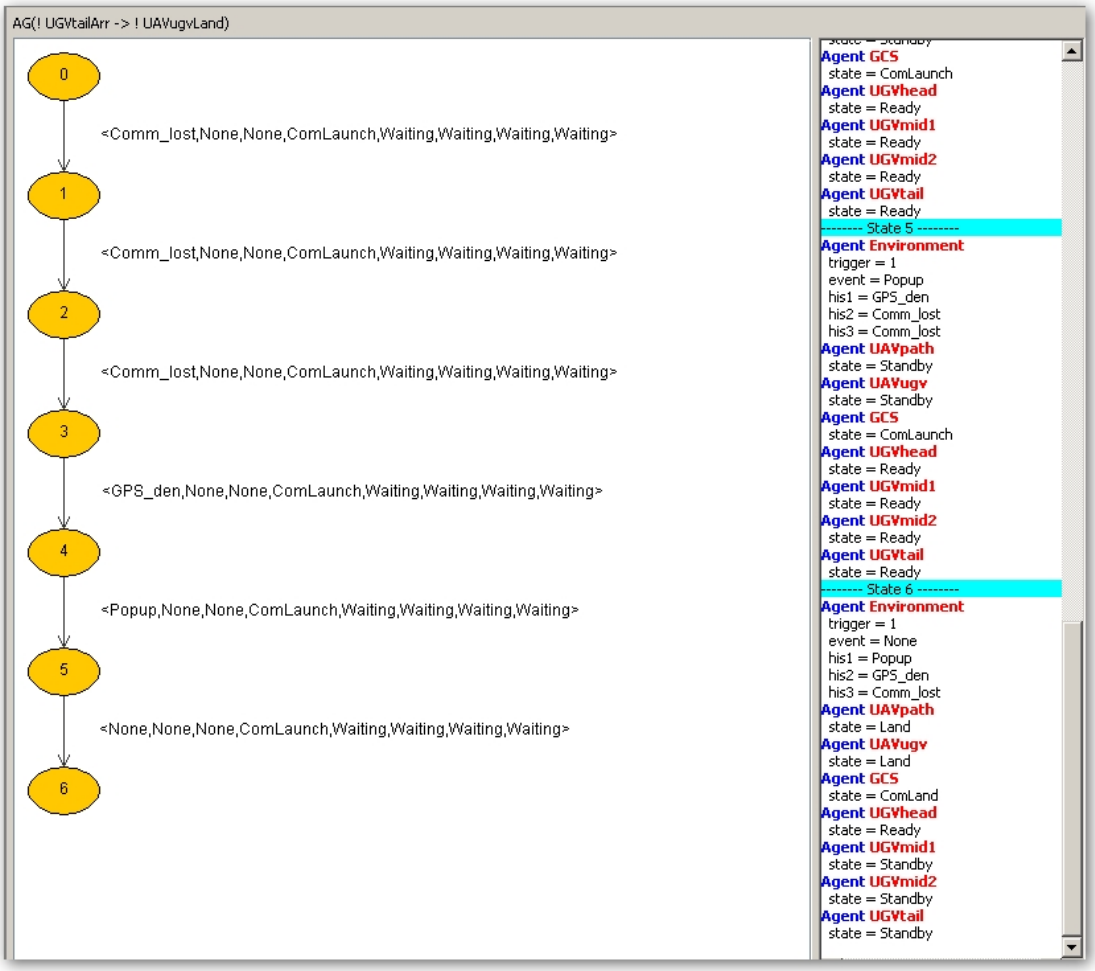

Figure 5.6: Counterexample of formula 9

There is one suspicious result of the verification, formula 9. As depicted in Figure 5.6, UAV 2 lands because communication loss is confirmed. For the safety reason, all the agents terminate the mission when they do not have valid communication channels. Therefore, in the counterexample, not only UAV 2 but also UAV 1 and the other UGVs are not performing the mission, either. Consequently, the safety property which was designed in the property modelling is not proved as *'true'*, but the counterexample stresses that all the agents are still safe. This counterexample derives importance of the priority among the properties in a design level. Specifically, UAV 2 must keep executing the monitoring mission for the safety of the UGVs but must interrupt it in emergency. That is to say, the property related with the safety for itself is more important than the property related with the safety for the other agent. The problem about the priority of the property is mentioned in the other research and is accompanied by not only unmanned systems but also manned systems [94].

The number of reachable states has been increased as much as five times compared with that of the previous chapter as shown in Figure 5.7. It is anticipated that this happened because the number of agents increased as many as more that two times. However, despite of this, the execution time is quite short and the memory used is ten mega bytes. It means that the CPU and RAM still have roomy capacity to deal with the verification of multi-agent systems with larger size. Because the extension of the number of agents has already been considered, more complex interaction within the agents will be dealt with in the following chapters. For example, each agent can have more states to represent more defined systems, in other words, to represent a higher level of granularity in the system.

```
execution time = 2
number of reachable states = 17793
BDD memory in use = 10327188
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 57692
**** CUDD non-modifiable parameters ****
Memory in use: 10327188
Peak number of nodes: 229950
Peak number of live nodes: 38672
Number of BDD variables: 89
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 1644515
Number of cache hits: 481819
Number of cache insertions: 1165318
Number of cache collisions: 432359
Number of cache deletions: 661911
Cache used slots = 98.35% (expected 74.77%)
Soft limit for cache size: 346112
Number of buckets in unique table: 86528
Used buckets in unique table: 47.16% (expected 46.96%)
Number of BDD and ADD nodes: 57030
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 28185
Number of dead ZDD nodes: 0
Total number of nodes allocated: 542988
Total number of nodes reclaimed: 964714
Garbage collections so far: 10
Time for garbage collection: 0.03 sec
Reorderings so far: 8
Time for reordering: 1.48 sec
```

Figure 5.7: BDD information of verification result

Chapter 6

Convoy mission by a group of UAVs and UGVs including fault problem

6.1 Scenario definition

Generally speaking, homogeneous multiple agents are considered to construct a redundant system. In this kind of system inclusive of redundancy, more than one agent executes the same task simultaneously to improve mission completion. Therefore, even though one of the multiple agents or more than one in a bigger system becomes faulty, the other agents can keep on performing the mission until achieving their ultimate goal. In this background, the scenario in Chapter 5 extends to include a fault-tolerant capability. First of all, the UGVs have the fault state individually like UAV 1. Then the mid 1, mid 2, and tail UGV are set to make their own decisions independently, i.e., their rules of behaviour are changed to a decentralized configuration from a leader-follower method. Consequently, the group of UGVs can be regarded as a redundant system to accomplish the convoy mission. Because UAV 2 already performs a role of redundancy for UAV 1, only the fault state is considered additionally.

6.2 Kripke modelling of multi-agent system

6.2.1 Kripke model for UAV 1

As described in Section 6.1, there is no modification from the previous model in the behaviours of UAV 1. Therefore, the Kripke model of UAV 1 is the same as the model in Section 5.2.1.

6.2.2 Kripke model for UAV 2

UAV 2 is planned to include the fault state in this scenario. Figure 6.1 represnts the reinforced Kripke model of UAV 2. The fault diagnosis is assumed to be done by itself.

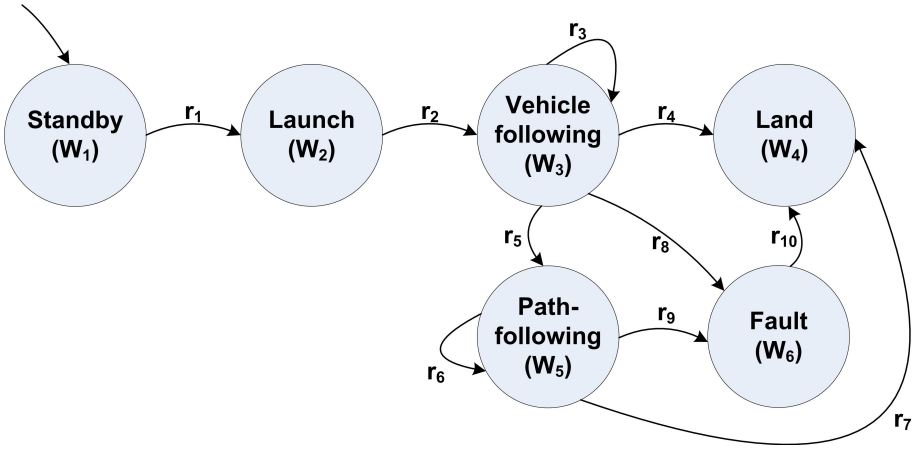

Figure 6.1: Kripke model for UAV 2

- $(W_1 - W_2)$: The transition happens if the head UGV is dispatched.
$$r_1 = \begin{cases} true & \text{if the head UGV is dispatched.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if UAV 2 has launched successfully.
$$r_2 = \begin{cases} true & \text{if UAV 2 has launched successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_3)$: The transition happens if UAV 2 performs its mission without any problem.
$$r_3 = \begin{cases} true & \text{if UAV 2 performs the vehicle-following.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_4, W_5 - W_4)$: The transition happens if UAV 2 receives the landing command, the UGVs have arrived, or the communication loss is confirmed.
$$r_4, r_7 = \begin{cases} true & \text{if UAV 2 receives the landing command} \\ & \text{, the UGVs have arrived,} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_5)$: The transition happens if UAV 1 reports its malfunction.
  $r_5 = \begin{cases} true & \text{if UAV 1 reports its malfunction.} \\ false & \text{otherwise} \end{cases}$
- $(W_5 - W_5)$: UAV 2 performs the path-following until it lands.
  $r_6 = \begin{cases} true & \text{if UAV 2 keeps performing the path-following.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_6, W_5 - W_6)$: The transition happens if UAV 2 self-detects its malfunctions.
  $r_8, r_9 = \begin{cases} true & \text{if UAV 2 detects its malfunctions.} \\ false & \text{otherwise} \end{cases}$
- $(W_6 - W_4)$: UAV 2 lands if it is faulty.
  $r_{10} = \begin{cases} true & \text{if UAV 2 is faulty.} \\ false & \text{otherwise} \end{cases}$

6.2.3 Kripke model for UGVs

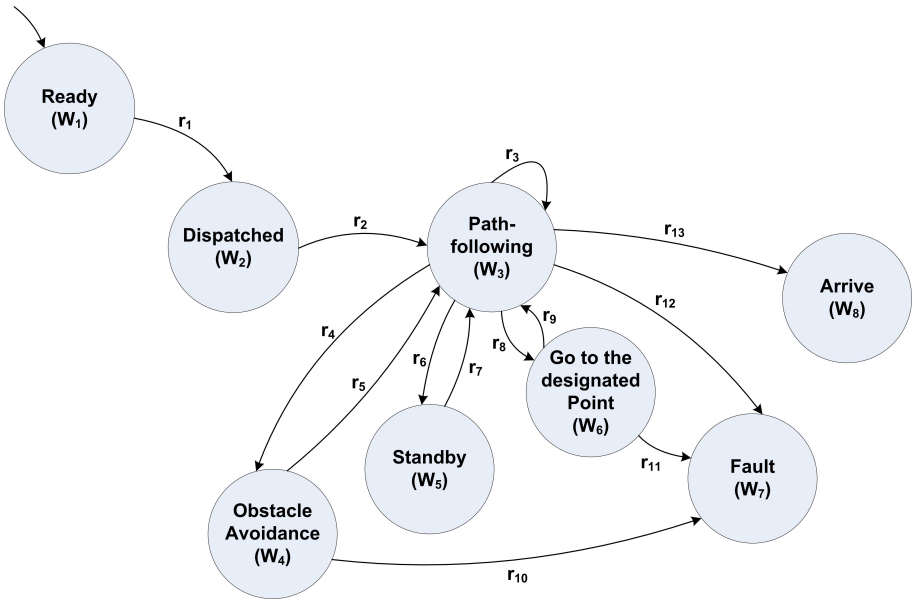

Figure 6.2: Kripke model for the UGV

Similarly to UAV 2, the Kripke model for UGVs has just one more state, the fault state, compared to the Kripke model in Chapter 5 as illustrated in Figure 6.2. The *accessibility relations* in Section 4.2.2 are the basis of the *accessibility relations* of the UGVs in this scenario, but the additional *accessibility relations* from the fault

state are considered. Moreover, to take account of realistic arrivals of the UGVs, different conditions are applied to the individual UGV for transition to the arrive state. These modifications are described as follows.

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.
  $r_1 = \begin{cases} true & \text{if the UGV receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$
- $(W_2 - W_3)$: The transition happens if the UGV has been dispatched successfully.
  $r_2 = \begin{cases} true & \text{if the UGV has been dispatched.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_3)$: The UGV keeps on the path-following state until it arrives at the destination unless there is a special event.
  $r_3 = \begin{cases} true & \text{if nothing happens and the mission keeps going on.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_4)$: The transition happens if the pop-up threats appear.
  $r_4 = \begin{cases} true & \text{if the pop-up threats appear.} \\ false & \text{otherwise} \end{cases}$
- $(W_4 - W_3)$: The transition happens if the UGV has avoided the threats successfully.
  $r_5 = \begin{cases} true & \text{if the UGV has avoided the threats.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_5)$: The transition happens if the main route has been damaged or the communication loss has been confirmed.
  $r_6 = \begin{cases} true & \text{if the route has been damaged} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$
- $(W_5 - W_3)$: The transition happens if the GCS re-plans the route.
  $r_7 = \begin{cases} true & \text{if the GCS re-plans the route.} \\ false & \text{otherwise} \end{cases}$
- $(W_3 - W_6)$: The transition happens if the UGV enters the GPS denial zone or an intermittent communication disorder happens.

$$r_8 = \begin{cases} true & \text{if the UGV enters the GPS denial zone} \\ & \text{or an intermittent communication disorder happens.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_3)$: The transition happens if the UGV gets out of the GPS denial zone and the communication has been restored.

$$r_9 = \begin{cases} true & \text{if the UGV gets out of the GPS denial zone} \\ & \text{and the communication has been restored.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_7, W_4 - W_7, W_6 - W_7)$: The transition happens if the UGV self-detects its fault.

$$r_{10}, r_{11}, r_{12} = \begin{cases} true & \text{if the UGV diagnoses its fault by itself.} \\ false & \text{otherwise} \end{cases}$$

Head UGV

- $(W_3 - W_8)$: The transition is executed if the head UGV arrives at the destination.

$$r_{13} = \begin{cases} true & \text{if the head UGV arrives at the destination.} \\ false & \text{otherwise} \end{cases}$$

Mid 1 UGV

- $(W_3 - W_8)$: The transition is executed if the mid 1 UGV arrives at the destination after the head UGV has arrived at the destination or been faulty.

$$r_{13} = \begin{cases} true & \text{if the mid 1 UGV arrives at the destination} \\ & \text{and the head UGV has arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

Mid 2 UGV

- $(W_3 - W_8)$: The transition is executed if the mid 2 UGV arrives at the destination after the head UGV and the mid 1 UGV have arrived at the destination or been faulty.

$$r_{13} = \begin{cases} true & \text{if the mid 2 UGV arrives at the destination} \\ & \text{and the head/mid 1 UGV have arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

Tail UGV

- ($W_3 - W_8$): The transition is executed if the tail UGV arrives at the destination after the other UGVs have arrived at the destination or been faulty.

$$r_{13} = \begin{cases} true & \text{if the tail UGV arrives at the destination} \\ & \text{and the other UGVs have arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

6.2.4 Kripke model for GCS

The Kripke model of the GCS is the same as the previous model as in the Section 4.2.3.

6.3 Modelling of properties to be verified

Two properties are verified in addition to the previous properties described in Section 5.3. One of them is related to redundancy management. Because the aim of using redundancy is to improve the possibility of the mission completion, the fault of a particular agent must not affect the execution of the others. To prove that this property is valid in the multi-agent system considered in this scenario, the contradictory property is investigated as *'If mid 1 or mid 2 UGV malfunctions, then the head UGV must not arrive at the destination.'* Another property is that *'If both UAV 1 and 2 are faulty, then the head UGV must not follow the path.'* This property can be regarded as the safety property of the UGV. UAV 1 performs the surveillance of the route which the UGVs travel along to inform the potential hazard. The main mission of UAV 2 is to monitor the group of UGVs to keep them safe as well. That is, if the UAVs have problems themselves, the UGVs may be exposed to the hazard directly. Therefore even if this is not considered in a design level, it is obvious that this property must be kept for the safety of the UGVs.

6.4 Model checking with MCMAS

6.4.1 MCMAS model

The *Agent Environment* is the same as before: has five variables expressing the events and the trigger variable to declare the permanent communication loss. In the *Agent UAV 2*, *Agent head UGV*, *Agent mid 1 UGV*, *Agent mid 2 UGV*, and *Agent tail UGV*, new state, action/protocol, and evaluation function related to the fault scenario are added. The modified *accessibility relations* due to realistic arrival explained in Section 6.2.3 are reflected in the *Agent UGVs* as well.

6.4.2 Verification result

The verification results from formula 1 to formula 11 are the same as those of Section 5.4.2 except formula 8. Formula 8 results in *'false'* which means that even though the head UGV goes to the standby mode by some events, the tail UGV does not always change its state to the standby state in all cases. Because the environmental events influence the decision-making of the agents globally, the result appears contradictory to this. Besides, two added formula also have *'false'*. In case of the formula 12, *'false'* means that malfunctions of some UGVs can not prevent the mission of the other UGVs from being completed. Therefore, these negative results can be regarded as correct behaviours. However, the *'false'* result of the formula 13 can be a serious problem because it indicates that the UGVs may keep on travelling along the route even if they lose their eye of the sky.

6.4.3 Analysis and discussion

In this chapter, the fault states of all the agents are integrated with the previous multi-agent system used in Chapter 5. These aim not only to consider more possible states in the scenario, but also to explore the scalability of MCMAS with a larger system in size. Moreover, beyond an occurrence of a fault, the fault-tolerant multi-agent system is constructed by changing the group-behaviour of UGVs from the leader-follower logic to the decentralized configuration.

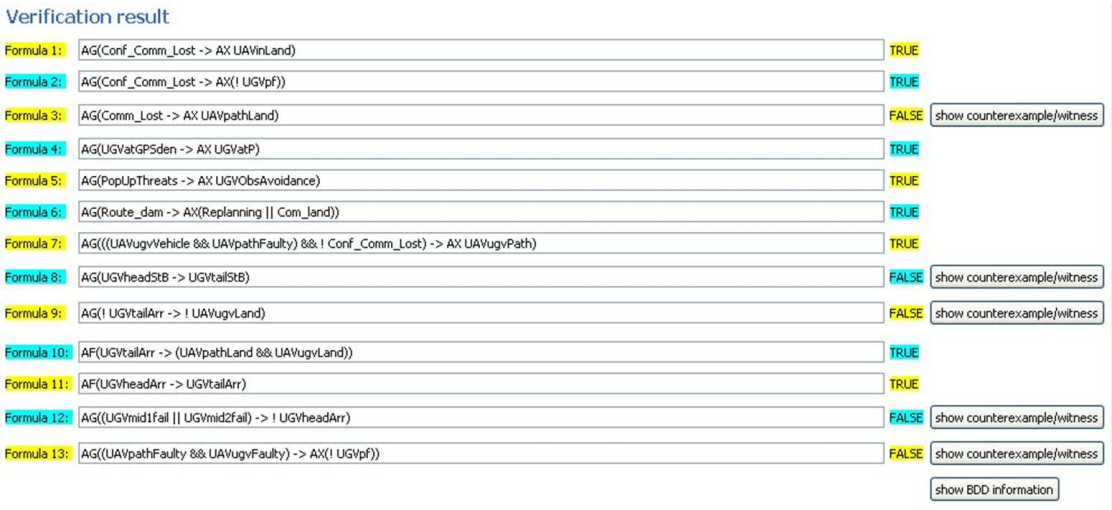

Figure 6.3: Verification Result

The intention of the design as described above is successfully verified with MCMAS. MCMAS has no difficulty to verify the extended multi-agent system and does not face the state space explosion. Furthermore, the properties represented as the formula 1 to 11, except 8, are proved to hold '*true*' in all possible cases. In other words, the additional behaviour and the modified configuration do not violate the basic properties that must keep in this multi-agent system. Furthermore, the result of formula 12 represents that the head UGV can arrive at the destination despite of failure of the mid 1 and 2 UGVs. According to this, it is successfully proved that the multi-agent system can be regarded as a fault-tolerant system.

The only unexpected result is that of the formula 8. There are two possible causes of this result which can be inferred from the modification of the system. The first one is the changed configuration of the UGV swarm. Another one is the additional fault state of the tail UGV. If the first one is right, it is contradictory to the design intention as mentioned in Section 6.4.2. Even though the group of UGVs changes their formation configuration from the leader-follower to the decentralized logic, the behaviours of each UGV are affected by the environmental events simultaneously. Therefore, if some event makes one UGV to change its state to the standby state, the other UGVs must change their state to the standby state as well. However, if the second possible cause is right, the '*false*' result of formula 8 is understandable. Since the tail UGV cannot react to the environmental events when it is faulty.

To solve the doubt about the result of the formula 8, its counterexample is investigated as depicted in Figure 6.4. The head UGV transits to the standby state because the trigger variable implying the permanent communication loss is turned to '1'. However, the tail UGV does not change the state according to the trigger variable because it is faulty. In common sense, the decision of the tail UGV is acceptable. Therefore, this result can be interpreted as an unexpected result from the additional state in the design level.

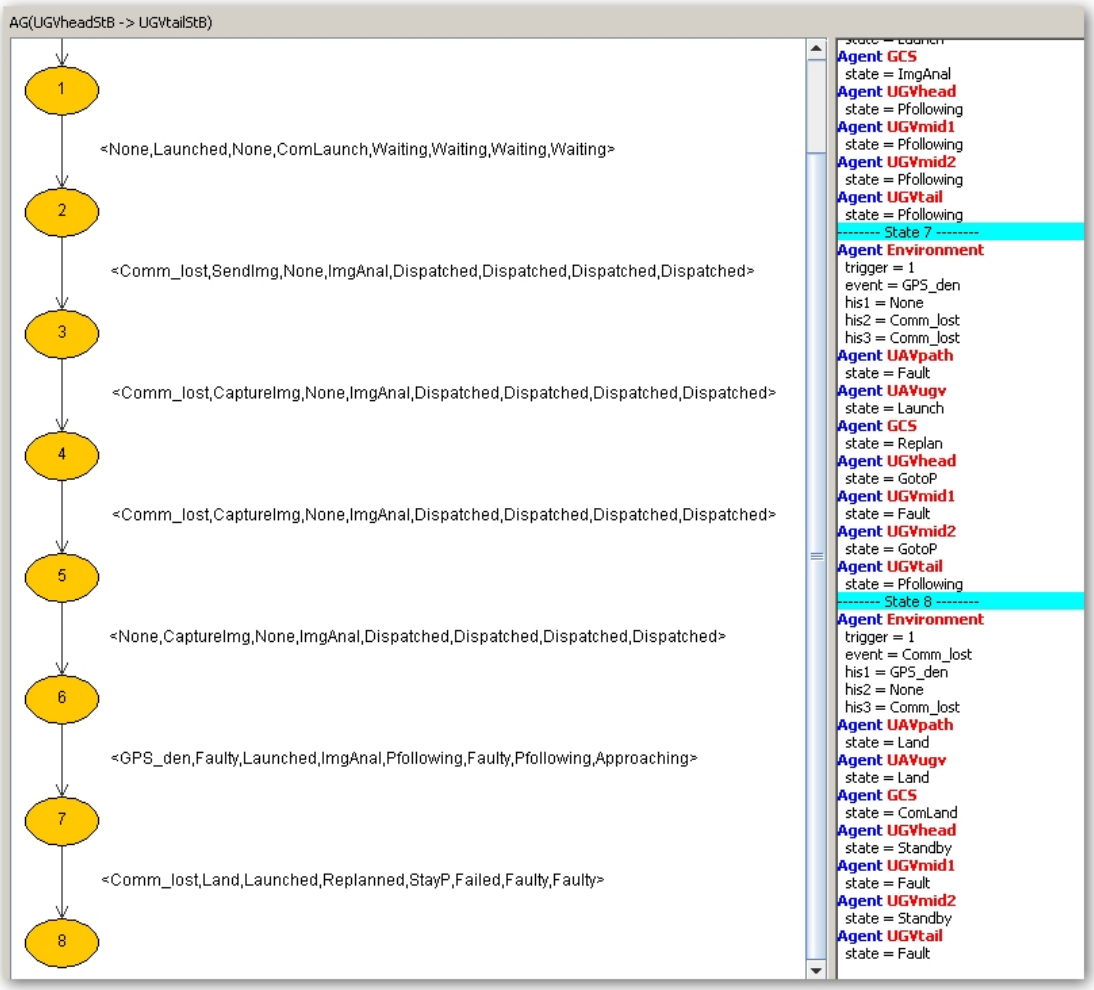

Figure 6.4: Counterexample of formula 8

The result of formula 13 is anticipated in the specification stage, differently from formula 8. The fault state is considered for the both UAVs but, proper decision-making algorithm for UGVs is not designed against the concurrent malfunctions of the UAVs. Figure 6.5 is the counterexample of the formula 13. The states of all UGVs are not connected with the malfunctions of the UAVs. As com-

mented in Section 6.3, this property must be valid in the system intuitively even though it has not been considered in the design level. By means of this result, the feasibility of model checking in the design level can be ascertained again. In summary, the results of model checking imply that model checking can provide a designer of autonomous systems with the clues of proactively hedging mistakes or missing algorithms.

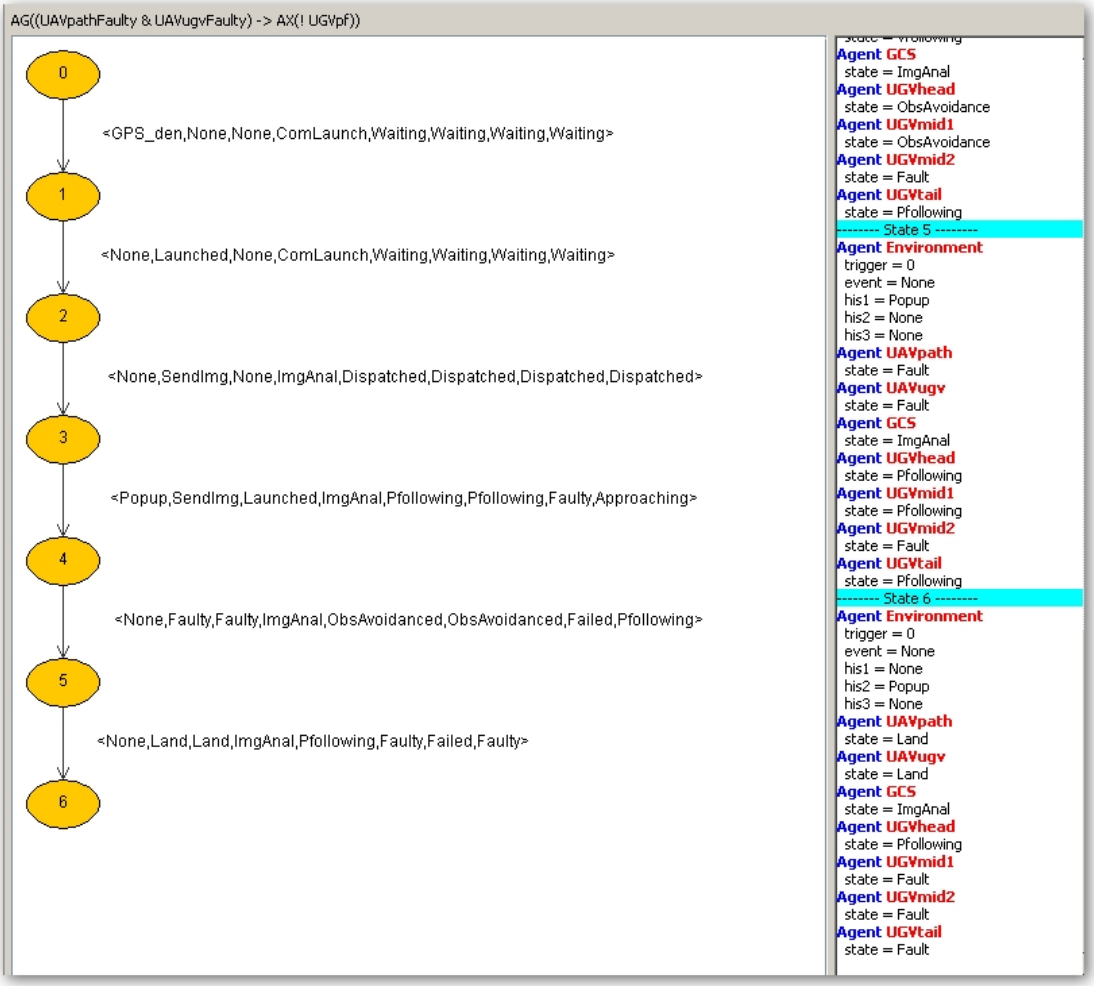

Figure 6.5: Counterexample of formula 13

The analysis of the above counterexamples will be reflected to the improved scenario in the next chapter. Addition to this, the next scenario will be more complex because Figure 6.6 reveals that MCMAS still has more potential to handle a larger size of complex multi-agent system. The execution time is around two seconds and the used memory is less than nine mega bytes. This is only 0.4 % of the total memory of the laptop which is used for verification.

```
execution time = 2
number of reachable states = 221503
BDD memory in use = 8895156
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 34520
**** CUDD non-modifiable parameters ****
Memory in use: 8895156
Peak number of nodes: 146146
Peak number of live nodes: 31664
Number of BDD variables: 93
Number of ZDD variables: 0
Number of cache entries: 262144
Number of cache look-ups: 776060
Number of cache hits: 278479
Number of cache insertions: 499800
Number of cache collisions: 100460
Number of cache deletions: 279739
Cache used slots = 84.58% (expected 76.62%)
Soft limit for cache size: 256000
Number of buckets in unique table: 64000
Used buckets in unique table: 83.62% (expected 83.52%)
Number of BDD and ADD nodes: 134125
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 112903
Number of dead ZDD nodes: 0
Total number of nodes allocated: 441624
Total number of nodes reclaimed: 746703
Garbage collections so far: 9
Time for garbage collection: 0.02 sec
Reorderings so far: 7
Time for reordering: 0.81 sec
```

Figure 6.6: BDD information of verification result

Chapter 7

Convoy mission by UAV and UGV swarms with communications relay

7.1 Scenario definition

A communication relay represents a station that relays messages between multiple points, so as to facilitate communications between them. Because it extends a coverage area of communication, there is an increasing interest in the implementation of communication relays for multi-agent systems [95]. Moreover, the communication relay can make indirect communication available in the case that a direct channel between agents is unhealthy. As a result, it can help maximise the survivability of agents. For that reason, the convoy mission scenario is now extended to include this communication relay behaviour. As depicted in Figure 7.1, communication among the agents are divided into three parts: channel A between the UAVs and GCS, channel B between the UGVs and GCS, and channel C between the UAVs and UGVs. The UAVs can use channels B and C to communicate with the GCS when they lose direct channel A, and similarly the UGVs can use channels A and C to communicate with the GCS when they lose channel B. In these cases, the UGVs and the UAVs play a role of communications rely agents to the GCS. Additionally, to reflect the analysis discussed in Section 6.4.3, the algorithm by which all the UGVs go to the standby mode when both of the UAVs are not in the sky is considered in this new scenario.

7.2 Kripke modelling of multi-agent system

7.2.1 Kripke model for UAV 1

To make *accessibility relations* simple, the path-following states are distinguished by the conditions of direct or indirect communications. Hence, UAV 1 has now six

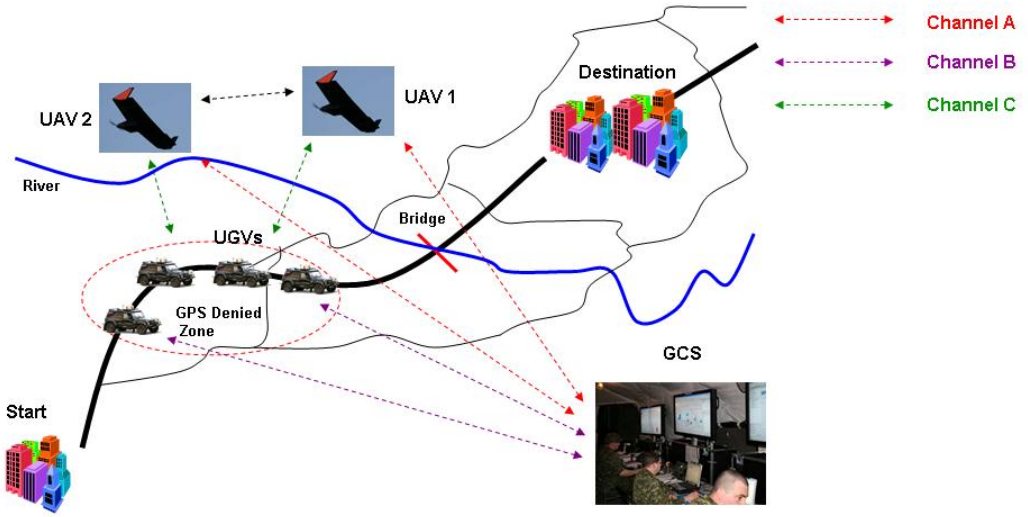

Figure 7.1: Convoy mission of multi-agent system using communication relay

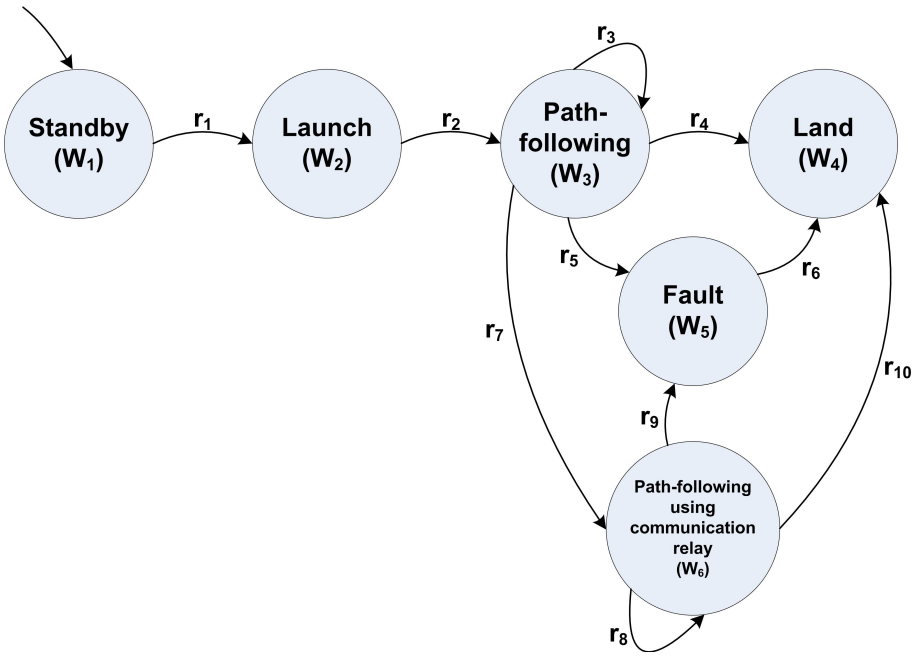

Figure 7.2: Kripke model for UAV 1

states as depicted in Figure 7.2: standby, launch, path-following, path-following using communications relay, fault, and land. The modified *accessibility relations* are expressed as follows.

- $(W_1 - W_2)$: The transition happens if UAV 1 receives the launching command.
$$r_1 = \begin{cases} true & \text{if UAV 1 receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if UAV 1 has launched successfully.
$$r_2 = \begin{cases} true & \text{if UAV 1 has launched successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_3)$: The transition happens if UAV 1 performs the path-following.
$$r_3 = \begin{cases} true & \text{if UAV 1 performs the path-following.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_4)$: The transition happens if UAV 1 receives the landing command, or the communication losses in channel A and B(or C) are confirmed.
$$r_4 = \begin{cases} true & \text{if UAV 1 receives the landing command,} \\ & \text{or the communication losses in channel A} \\ & \text{and B (or C) are confirmed.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_5, W_6 - W_5)$: The transition happens if UAV 1 self-detects its malfunctions.
$$r_5, r_9 = \begin{cases} true & \text{if UAV 1 detects its malfunctions.} \\ false & \text{otherwise} \end{cases}$$
- $(W_5 - W_4)$: UAV 1 lands if it is faulty.
$$r_6 = \begin{cases} true & \text{if UAV 1 is faulty.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_6)$: The transition happens if the communication loss in channel A is confirmed and the other communication channels are healthy.
$$r_7 = \begin{cases} true & \text{if the communication loss in channel A is confirmed} \\ & \text{and the other communication channels are healthy.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_6)$: The transition happens if UAV 1 performs the path-following by using the indirect communication channel.

$$r_8 = \begin{cases} true & \text{if UAV 1 performs the path-following} \\ & \text{with the indirect communication channel.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_4)$: The transition happens if UAV 1 loses the indirect communication channel.

$$r_{10} = \begin{cases} true & \text{if UAV 1 loses the indirect communication channel.} \\ false & \text{otherwise} \end{cases}$$

7.2.2 Kripke model for UAV 2

Compared with the UAV 1, the Kripke model for UAV 2 has been modified to be more complex as shown in Figure 7.3. There are eight states: standby, launch, vehicle-following, vehicle-following using communications relay, path-following, path-following using communications relay, fault, and land.

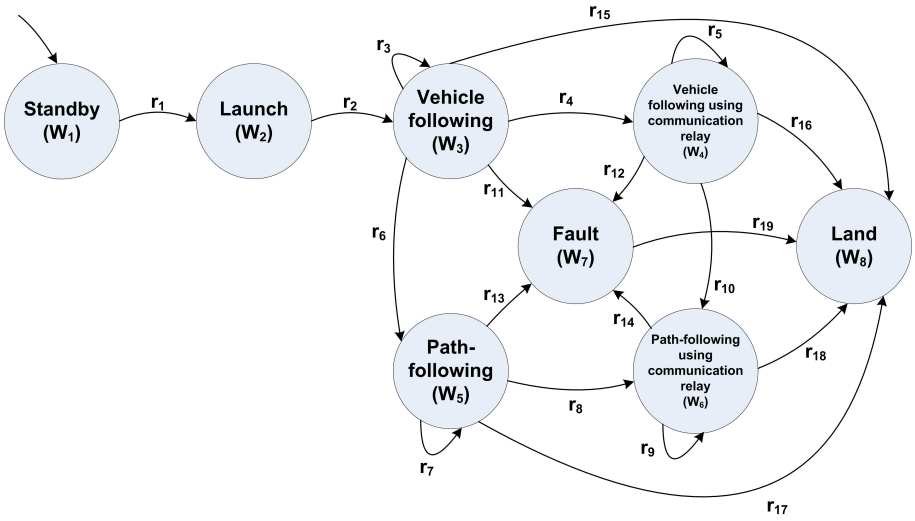

Figure 7.3: Kripke model for UAV 2

The modified *accessibility relations* can be represented as follows.

- $(W_1 - W_2)$: The transition happens if the head UGV is dispatched.

$$r_1 = \begin{cases} true & \text{if the head UGV is dispatched.} \\ false & \text{otherwise} \end{cases}$$

- $(W_2 - W_3)$: The transition happens if UAV 2 has launched successfully.

$$r_2 = \begin{cases} true & \text{if UAV 2 has launched successfully.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_3)$: The transition happens if UAV 2 performs its mission without any problem.

$$r_3 = \begin{cases} true & \text{if UAV 2 performs the vehicle-following.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_4)$: The transition happens if the communication loss in channel B is confirmed while UAV 2 performs vehicle-following.

$$r_4 = \begin{cases} true & \text{if the channel B is confirmed to be lost} \\ & \text{while UAV 2 performs vehicle-following.} \\ false & \text{otherwise} \end{cases}$$

- $(W_4 - W_4)$: The transition happens if UAV 2 performs its mission using the indirect communication channel.

$$r_5 = \begin{cases} true & \text{if UAV 2 performs the vehicle-following.} \\ & \text{using the indirect communication channel.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_5)$: The transition happens if UAV 1 reports its malfunction.

$$r_6 = \begin{cases} true & \text{if UAV 1 reports its malfunction.} \\ false & \text{otherwise} \end{cases}$$

- $(W_5 - W_5)$: UAV 2 keeps path-following unless something happens.

$$r_7 = \begin{cases} true & \text{if UAV 2 keeps performing the path-following} \\ & \text{unless something happens.} \\ false & \text{otherwise} \end{cases}$$

- $(W_5 - W_6)$: The transition happens if the communication loss in channel B is confirmed while UAV 2 performs path-following.

$$r_8 = \begin{cases} true & \text{if the channel B is confirmed to be lost} \\ & \text{while UAV 2 performs path-following.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_6)$: The transition happens if UAV 2 performs its mission using the indirect communication channel.

$$r_9 = \begin{cases} true & \text{if UAV 2 performs the path-following.} \\ & \text{using the indirect communication channel.} \\ false & \text{otherwise} \end{cases}$$

- $(W_4 - W_6)$: The transition happens if UAV 1 reports its malfunction while the indirect communication channel is used.

$$r_{10} = \begin{cases} true & \text{if UAV 1 reports its malfunction.} \\ & \text{while the indirect channel is used.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_7, W_4 - W_7, W_5 - W_7, W_6 - W_7)$: The transition happens if UAV 2 detects its malfunction.

$$r_{11}, r_{12}, r_{13}, r_{14} = \begin{cases} true & \text{if UAV 2 detects its malfunction.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_8, W_4 - W_8, W_5 - W_8, W_6 - W_8)$: The transition happens if UAV 2 receives the landing command, the UGVs have arrived, or the communication loss is confirmed in both of direct and indirect one.

$$r_{15}, r_{16}, r_{17}, r_{18} = \begin{cases} true & \text{if UAV 2 receives the landing command} \\ & \text{, the UGVs have arrived, or both of the direct} \\ & \text{and indirect communication losses are confirmed.} \\ false & \text{otherwise} \end{cases}$$

- $(W_7 - W_8)$: UAV 2 lands if it is faulty.

$$r_{19} = \begin{cases} true & \text{if UAV 2 is faulty.} \\ false & \text{otherwise} \end{cases}$$

7.2.3 Kripke model for UGVs

The added *possible world* of the UGVs is the only state in which the UGVs performs the path-following with the indirect communication channel. Differently from the state, *accessibility relations* become more complex as shown in Figure 7.4.

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.

$$r_1 = \begin{cases} true & \text{if the UGV receives the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$

- $(W_2 - W_3)$: The transition happens if the UGV has been dispatched successfully.

$$r_2 = \begin{cases} true & \text{if the UGV has been dispatched.} \\ false & \text{otherwise} \end{cases}$$

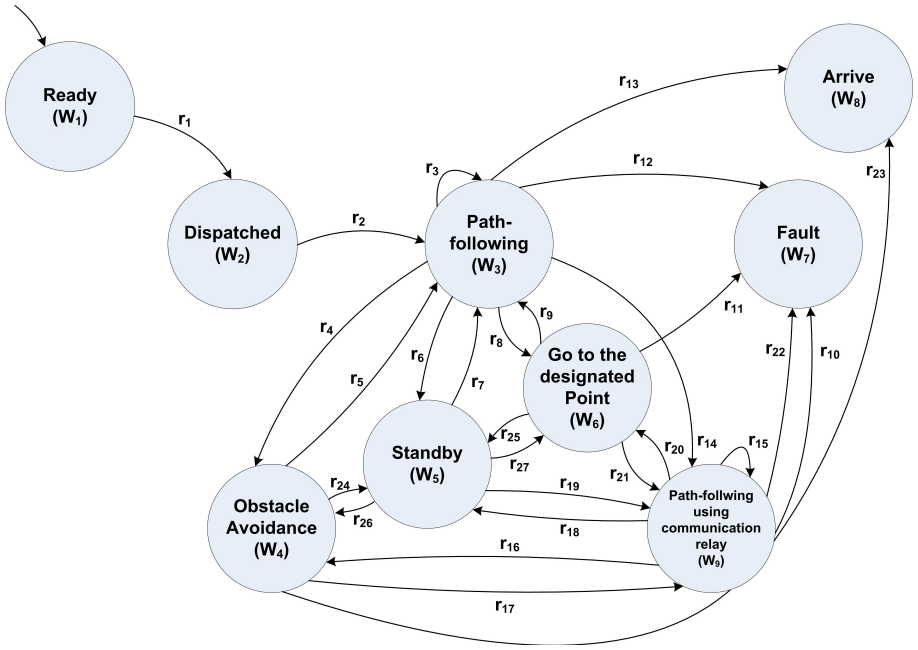

Figure 7.4: Kripke model for the UGV

- $(W_3 - W_3)$: The UGV keeps on the path-following state until it arrives at the destination unless there is a special event.

$$r_3 = \begin{cases} true & \text{if nothing happens and the mission keeps going on.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_4, W_9 - W_4)$: The transition happens if the pop-up threats appear.

$$r_4, r_{16} = \begin{cases} true & \text{if the pop-up threats appear.} \\ false & \text{otherwise} \end{cases}$$

- $(W_4 - W_3, W_4 - W_9)$: The transition happens if the UGV has avoided the threats successfully.

$$r_5, r_{17} = \begin{cases} true & \text{if the UGV has avoided the threats.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_5, W_4 - W_5, W_6 - W_5, W_9 - W_5)$: The transition happens if the main route has been damaged, both of the UAVs have landed, or the communication loss has been confirmed.

$$r_6, r_{18}, r_{24}, r_{25} = \begin{cases} true & \text{if the route has been damaged,} \\ & \text{both of the UAVs have landed,} \\ & \text{or the communication loss is confirmed.} \\ false & \text{otherwise} \end{cases}$$

- $(W_5 - W_3, W_5 - W_4, W_5 - W_6, W_5 - W_9)$: The transition happens if the GCS re-plans the route.

$$r_7, r_{19}, r_{26}, r_{27} = \begin{cases} true & \text{if the GCS re-plans the route.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_6, W_9 - W_6)$: The transition happens if the UGV enters the GPS denial zone or an intermittent communication disorder happens.

$$r_8, r_{20} = \begin{cases} true & \text{if the UGV enters the GPS denial zone} \\ & \text{or an intermittent communication disorder happens.} \\ false & \text{otherwise} \end{cases}$$

- $(W_6 - W_3, W_6 - W_9)$: The transition happens if the UGV gets out of the GPS denial zone and the communication has been restored.

$$r_9, r_{21} = \begin{cases} true & \text{if the UGV gets out of the GPS denial zone} \\ & \text{and the communication has been restored.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_7, W_4 - W_7, W_6 - W_7, W_9 - W_7)$: The transition happens if the UGV self-detects its fault.

$$r_{10}, r_{11}, r_{12}, r_{22} = \begin{cases} true & \text{if the UGV diagnoses its fault by itself.} \\ false & \text{otherwise} \end{cases}$$

- $(W_3 - W_9)$: The transition happens if the communication channel B is lost permanently, and the other channels are healthy.

$$r_{14} = \begin{cases} true & \text{if the communication channel B is lost permanently} \\ & \text{, and the other channels are healthy.} \\ false & \text{otherwise} \end{cases}$$

- $(W_9 - W_9)$: The UGV keeps on path-following until it arrives at the destination using the indirect communication channel.

$$r_{15} = \begin{cases} true & \text{if the UGVs keep on executing mission} \\ & \text{using the indirect communication channel.} \\ false & \text{otherwise} \end{cases}$$

Head UGV

- $(W_3 - W_8, W_9 - W_8)$: The transition is executed if the head UGV arrives at the destination.

$$r_{13}, r_{23} = \begin{cases} true & \text{if the head UGV arrives at the destination.} \\ false & \text{otherwise} \end{cases}$$

Mid 1 UGV

- $(W_3 - W_8, W_9 - W_8)$: The transition is executed if the mid 1 UGV arrives at the destination after the head UGV has arrived at the destination or been faulty.

$$r_{13}, r_{23} = \begin{cases} true & \text{if the mid 1 UGV arrives at the destination} \\ & \text{and the head UGV has arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

Mid 2 UGV

- $(W_3 - W_8, W_9 - W_8)$: The transition is executed if the mid 2 UGV arrives at the destination after the head UGV and the mid 1 UGV have arrived at the destination or been faulty.

$$r_{13}, r_{23} = \begin{cases} true & \text{if the mid 2 UGV arrives at the destination} \\ & \text{and the head/mid 1 UGV have arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

Tail UGV

- $(W_3 - W_8, W_9 - W_8)$: The transition is executed if the tail UGV arrives at the destination after the other UGVs have arrived at the destination or been faulty.

$$r_{13}, r_{23} = \begin{cases} true & \text{if the tail UGV arrives at the destination} \\ & \text{and the other UGVs have arrived at the destination} \\ & \text{or been faulty.} \\ false & \text{otherwise} \end{cases}$$

7.2.4 Kripke model for GCS

In the same way as the previous chapters, the GCS does not have any change in the *possible worlds*. However, because there is the indirect communications channel between the UAVs and GCS, the GCS commands the UAVs to land when they lose both direct and indirect communication channels. This modification is reflected in *accessibility relations* as follows:

- $(W_1 - W_2)$: The transition happens if the GCS sends the launching command.
$$r_1 = \begin{cases} true & \text{if the GCS sends the launching command successfully.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_2)$: The state maintains if the GCS performs the image analysis.
$$r_2 = \begin{cases} true & \text{if the GCS performs the image analysis.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_3)$: The transition happens if the GCS decides the re-planning.
$$r_3 = \begin{cases} true & \text{if the GCS decides the re-planning} \\ & \text{because of GPS denial or damaged route.} \\ false & \text{otherwise} \end{cases}$$
- $(W_3 - W_2)$: The transition happens if the GCS has finished the re-planning.
$$r_4 = \begin{cases} true & \text{if the GCS has finished the re-planning.} \\ false & \text{otherwise} \end{cases}$$
- $(W_2 - W_4)$: The transition happens if the mission has been completed or the communication loss is confirmed in both direct and indirect channels.
$$r_5 = \begin{cases} true & \text{if the mission has been completed} \\ & \text{or the communication loss is confirmed} \\ & \text{in both direct and indirect channels.} \\ false & \text{otherwise} \end{cases}$$

7.3 Modelling of properties to be verified

All the properties defined in Section 6.3 are employed again with some modification. According to the analysis about the result of formula 8 in the previous chapter, formula 8 is changed to be more reasonable as *'If the head UGV turns to the standby state, then the tail UGV must turn to the standby state unless it is faulty.'* Moreover, formula 13 is reinforced to check whether all the UGVs do not keep going along the path when both UAV 1 and 2 are faulty.

7.4 Model checking with MCMAS

7.4.1 MCMAS model

The most remarkable change in this MCMAS model compared with the previous models is that events are divided into two groups: *Comm_lost* and the others. To use trigger variables for the confirmation of the communication loss as global variables, *Agent Environment* consists of the states of three communication channels. There are nine states representing valid communication channels: No, chA, chB, chC, chAB, chAC, chBC, chABC. According to these states, each channel is decided to be healthy or not, and its temporary variable accumulates its history during three sequences. Then, three trigger variables announce the confirmation of the communication loss using the history. The remained *None*, *GPS_den*, *Route_dam*, and *Popup* are constructed in a new agent part, *Agent Env*, and have the same structure as the previous *Agent Environment*. Also, the rest *Agents* are built based on the Kripke model in Section 7.2

7.4.2 Verification result

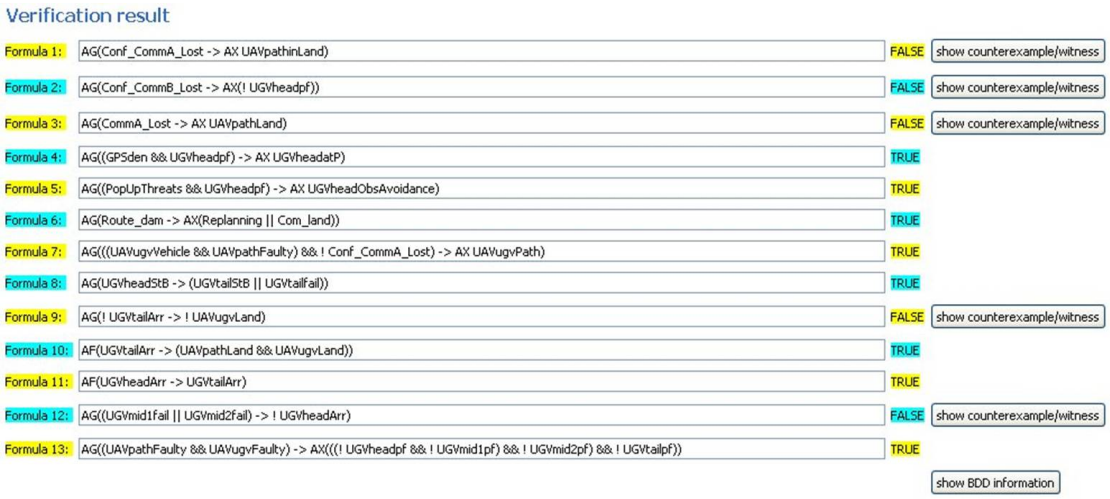

Figure 7.5: Verification Result

First of all, the formula 1 and 2 have *'false'* results distinctively from the verification result in Section 6.4.2. They can be interpreted as that the UAVs and

UGVs do not terminate their mission when they lose the direct communication channels with the GCS. In case of the formula 8, as explained in Section 7.3, the property is changed to fit to the modified system including the fault problem. As a result, *'false'* in Figure 6.3 is now changed to *'true'*. The decision-making algorithms related with the property 13 were reflected in this scenario as well. Therefore, it is now verified that the UGVs stop travelling when they lose the eye of the sky.

7.4.3 Analysis and discussion

The key behaviour added in this chapter is that of communications relay. The UAVs and the UGVs are designed to use the indirect communication channel when they lose the direct communication channel with the GCS rather they terminate their missions. These additional behaviours aim to increase the probability of the mission completion. As described in Section 7.4.2, the UAVs and the UGVs do not terminate their missions pursuant to additional decision-making algorithms using communications relay. It shows that the aim of modifying the previous system has been achieved successfully.

To assure that the UAVs and UGVs use the indirect communication channels when they can not use the direct channels permanently, the counterexamples of the formula 1 and 2 are presented as follows.

In Figure 7.6, the *UAVpath* (UAV 1) switched its state from *Pfollowing* (Path-following) to *PFcommrelay* (Path-following using communication relay) when the trigger variable of the channel A turned to '1'. As well as UAV 1, the state of the *UAVugv* (UAV 2) changed from *Vfollowing* (Vehicle-following) to *VFcommrelay* (Vehicle-following using communication relay) simultaneously. Figure 7.7 shows that when the trigger variable of the channel B was turned on, the head and tail UGVs switched their states from *Pfollowing* to *PFcommrelay*. Because the mid 1 and mid 2 UGVs were in the fault state, they did not need to be considered.

In summary at last, the heterogeneous multi-agent system in this chapter has the decision-making behaviours described as follows:

- The UAVs and the UGVs can decide to use the indirect communication channels when they lose their direct communication channels with the GCS

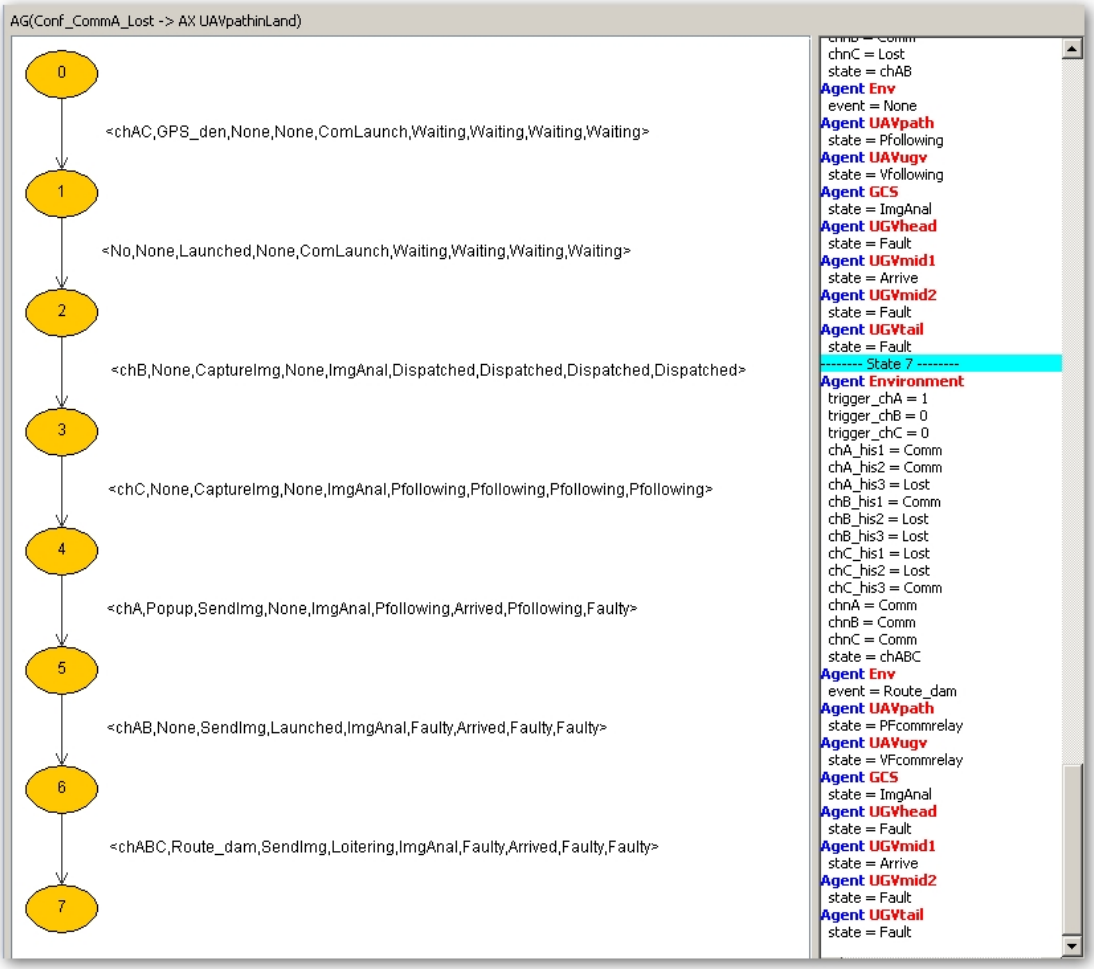

Figure 7.6: Counterexample of formulae 1

(proved by the counterexamples of the formula 1 and 2)

- UAV 1 maintains its behaviour despite of an intermittent communication loss to improve the mission completion (proved by the formula 3)

- The head UGV can make proper decision when it encounters with the GPS-denying area or pop-up threats. (proved by the formula 4 and 5)

- The GCS can re-plan the mission or decide to terminate the mission when the main route is damaged. (proved by the formula 6)

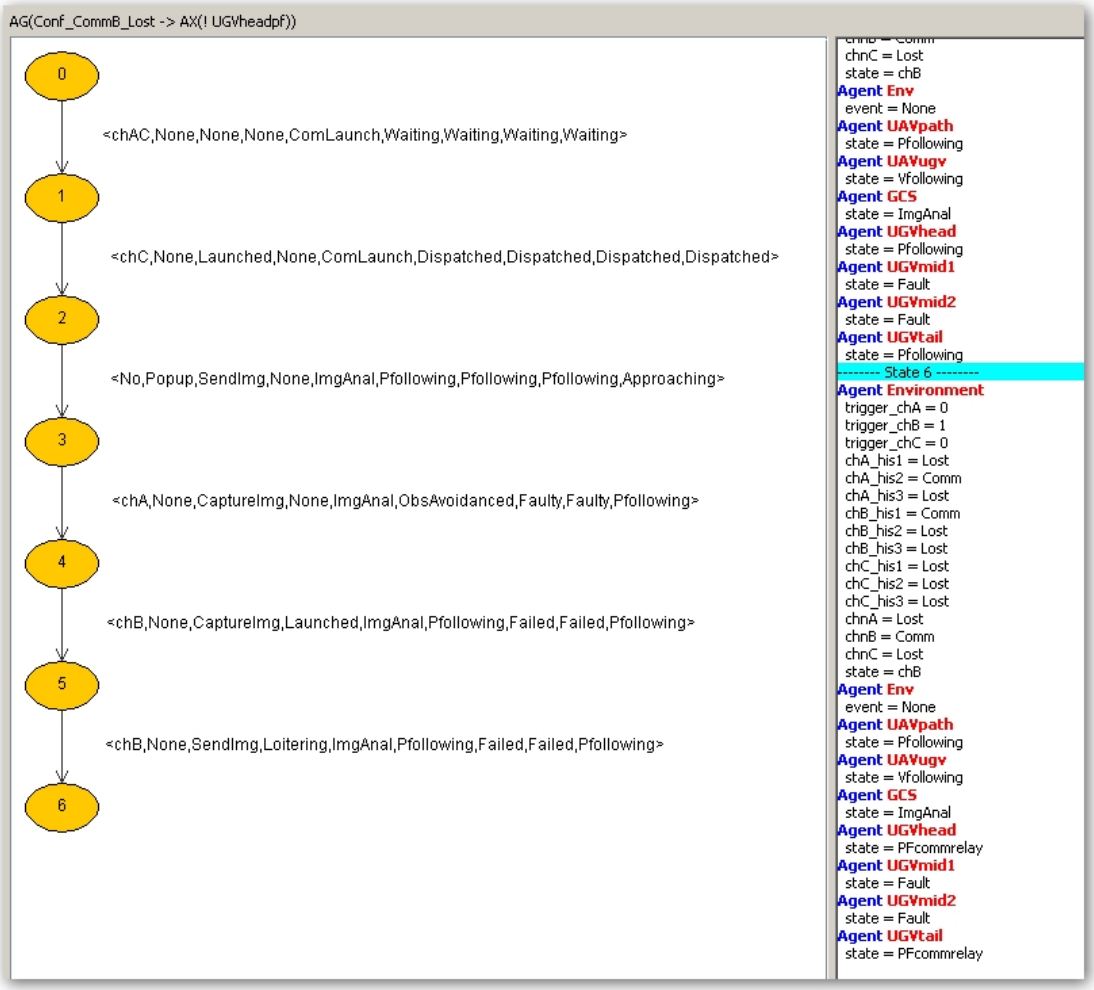

Figure 7.7: Counterexample of formulae 2

- UAV 2 can take the role of UAV 1 when UAV 1 is faulty. (proved by the formula 7)

- The UGVs can make correct decision against the events for themselves unless they are not faulty (proved by the formula 8)

- UAV 2 can decide to land when it lost the communication channels even though the UGV has not finished the mission according to the high priority of the property. (proved by the formula 9)

- The UAVs can finish their mission if the tail UGV completes its mission.

(proved by the formula 10)

- The tail UGV can arrive at the destination at last if the head UGV arrives at the destination. (proved by the formula 11)
- The head UGV can arrive at the destination independently though the mid 1 UGV or the mid 2 UGV is faulty. (proved by the formula 12)
- The UGV swarm decide to stop path-following when they lose the convoy by both of the UAVs by safety requirement. (proved by the formula 13)

As the scenario of this chapter became more complex, MCMAS explored more reachable states as shown in Figure 7.8. Actually, twenty millions of reachable states in Figure 7.8 are difficult to be handled by numerical simulations. Also, the MCMAS capability to accommodate even nine agents is remarkable compared with the other model checkers. These facts represent the superior scalability of the MCMAS against the state-space explosion problem which many other model checking tools face. As a result, the proposed scenario considered in this chapter ascertains the potential applicability of the MCAS for the verification of multi-agent systems composed of many number of agents.

```
execution time = 66
number of reachable states = 2.80388e+08
BDD memory in use = 130215892
**** CUDD modifiable parameters ****
Hard limit for cache size: 5592405
Cache hit threshold for resizing: 30%
Garbage collection enabled: yes
Limit for fast unique table growth: 3355443
Maximum number of variables sifted per reordering: 1000
Maximum number of variable swaps per reordering: 2000000
Maximum growth while sifting a variable: 1.2
Dynamic reordering of BDDs enabled: no
Default BDD reordering method: 4
Dynamic reordering of ZDDs enabled: no
Default ZDD reordering method: 4
Realignment of ZDDs to BDDs enabled: no
Realignment of BDDs to ZDDs enabled: no
Dead nodes counted in triggering reordering: no
Group checking criterion: 7
Recombination threshold: 0
Symmetry violation threshold: 0
Arc violation threshold: 0
GA population size: 0
Number of crossovers for GA: 0
Next reordering threshold: 895769
**** CUDD non-modifiable parameters ****
Memory in use: 130215892
Peak number of nodes: 3496262
Peak number of live nodes: 638666
Number of BDD variables: 117
Number of ZDD variables: 0
Number of cache entries: 4194304
Number of cache look-ups: 28937847
Number of cache hits: 8997626
Number of cache insertions: 19976126
Number of cache collisions: 15181014
Number of cache deletions: 2335434
Cache used slots = 58.64% (expected 53.53%)
Soft limit for cache size: 5006336
Number of buckets in unique table: 1251584
Used buckets in unique table: 90.61% (expected 90.52%)
Number of BDD and ADD nodes: 3495710
Number of ZDD nodes: 0
Number of dead BDD and ADD nodes: 3080820
Number of dead ZDD nodes: 0
Total number of nodes allocated: 7761101
Total number of nodes reclaimed: 22760757
Garbage collections so far: 18
Time for garbage collection: 0.37 sec
Reorderings so far: 15
Time for reordering: 54.47 sec
```

Figure 7.8: BDD information of verification result

Chapter 8

Conclusions and Future Work

8.1 Summary

This thesis tackled the problems of model checking using MCMAS for verifying the decision making behaviours of multi-agent autonomous system. The Kripke model and temporal logic were used to construct the model and specification of the multi-agent system of interest. Chapter 2 described formal methods with three processes: formal modelling including the Kripke model, verifiable logics including a temporal logic, and model checking tools including the MCMAS. In Chapter 3, a simple surveillance scenario was captured from MoD Grand Challenge where a MAV flied following waypoints and captured the images of potential threats and at the same time a GCS received the images from the MAV and analyses them. From the model checking result with this scenario, it was found that mission completion can not be guaranteed because of communication loss. To supplement that system, the modified scenario with adding a substitute MAV was introduced in the same chapter. Additionally, the concept of confirmation time was used to decide whether the communication is lost permanently or not. The multi-agent system dealt with in Chapter 4 was composed of a GCS, a UAV loitering in the path ahead of a UGV, and a UGV transporting the resources across the battlefield. Its scenario focused on four events that could potentially deter mission completion. Moreover, as aforementioned, the communication loss was taken into account to affect the termination of mission only in the case that it is confirmed by the confirmation time. The decision making behaviours were specified and their feasibility were verified using MCMAS. In Chapter 5, the multi-agent system was extended to a larger system possessing two UAVs, four UGVs, and a GCS to improve mission completion. Moreover, Chapter 6 and Chapter 7 showed a more complicated multi-agent system with fault-tolerant and communication relay behaviours, respectively. The Kripke model, the computational tree logic, and the MCMAS were also used to abstract the multi-agent system and their properties and to perform model checking automatically. Figure 8.1 represents the extension of system in Chapter 3, and Figure 8.2 shows the overview of system extension in

the convoy scenario. For the convenience, each multi-agent system considered in Chapters 3 to 7 is named as Sys 1 to 6.

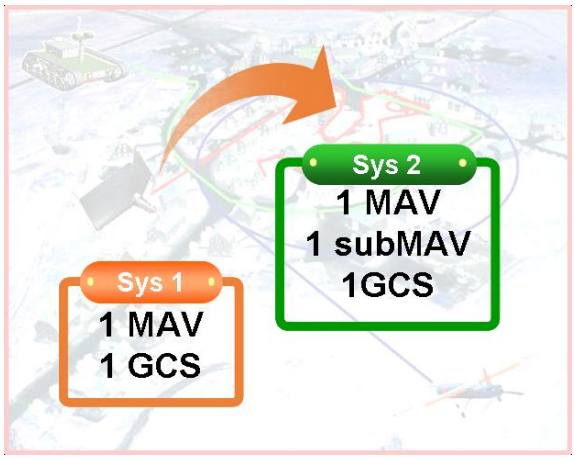

Figure 8.1: System extension of surveillance mission scenario

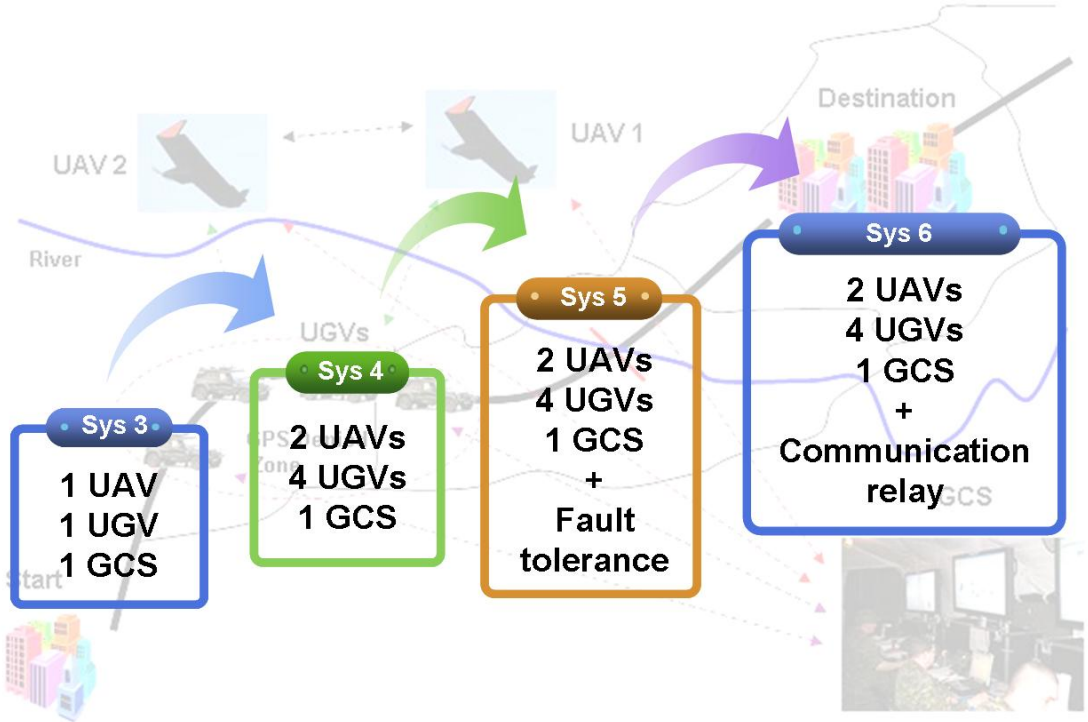

Figure 8.2: System extension of convoy mission scenario

8.2 Discussions

This thesis investigated how model checking can be used in the design-level of operating a heterogeneous multi-agent system. As the multi-agent system was modified accordingly by reflecting the previous verification result, each chapter checked the counterexamples of *'false'* results and successfully found the mistakes that might be ignored by a system designer. This whole process demonstrates how model checking can be used for verifying the decision-making logics of multi-agent systems in the design-level. Moreover, by extending scenario without verifying some behaviours already verified before, model checking can be considered as an efficient and practical design tool for verification of autonomous systems.

Figure 8.3: Comparison of BDD data

Figure 8.3 illustrates the comparison of the most important BDD data in each chapter excluding Sys 1. The execution time never exceeded 2 minutes even

though the number of reachable states approached 20 million. Because Sys 6 was composed of nine agents including *Agent Environment* and *Agent Env*, an execution time of around 1 minute can be remarkably short comparing with research using the other model checkers. For example, there is a study using SMV to verify UAVs which perform road network monitoring [96]. Table 5.3-5.6 in this thesis demonstrate the effect of number of UAVs/roads on the execution time and reachable states. As described in Table 5.4 and 5.5, SMV consumed 79 hours and 33 minutes to verify the multi-agent system consisting of four UAVs by exploring 1.25809 million states. In other words, MCMAS can explore almost seventeen times reachable states only consuming 0.02% execution time.

Moreover, MCMAS shows off its scalability in Figure 8.4. The graphs represent histories of the execution time, reachable states, and memory in use. The used memory in each scenario has similar trace with the execution time. Furthermore, the potential scalability of MCMAS can be inferred from the much lower level of memory in use than the RAM capacity in the graph.

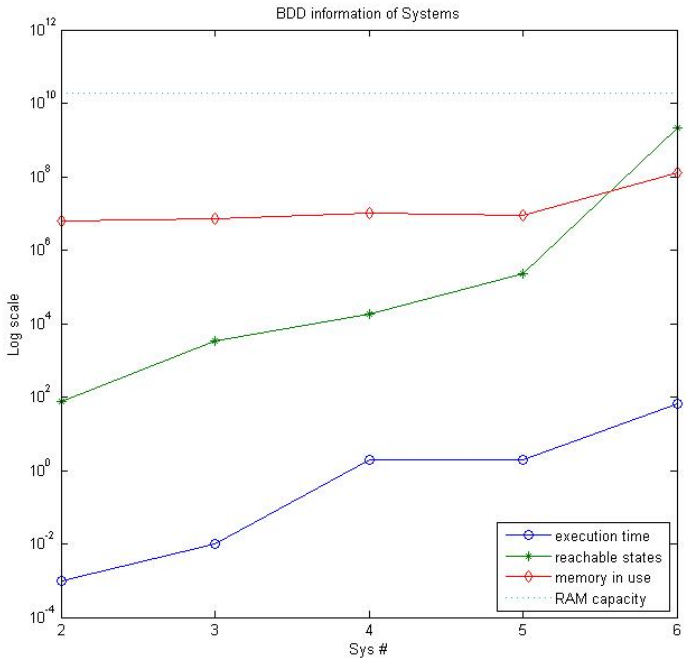

Figure 8.4: Comparison of memory in use

In conclusion, the result of this thesis clearly affirmed that the MCMAS handled a multi-agent system easily and efficiently in computation. Furthermore, as this

study has not yet experienced the state-space explosion despite of the expansion of multi-agent system in size, it is anticipated that more agents can be handled by the MCMAS.

8.3 Future Work

As mentioned in Sesction 8.2, the extended multi-agent system possessing ten or more agents can be dealt with by the MCMAS verification. For instance, the extended system from the final scenario in Chapter 7 can be constructed: a troop of UAVs monitor different routes on the ground assigned for an individual UGV and send the state of routes to the GCS, respectively. The GCS integrates the information of the routes and decides which route the UGVs can drive along with. Through this extension, the scenario will become more complex but absolutely practical. Frankly speaking, the most difficult part in this study was system modelling. It is not easy to capture all the behaviours of a heterogeneous multi-agent system intuitively. From that reason, the translation program which transforms a system design to a model checker's input language is desired to reduce the effort/mistake of a designer. Also, C, C++, or MATLAB stateflow are widely used to construct the decision-making algorithms in autonomous system in a recent time. If a translation program from the C, C++, or MATLAB to the MCMAS is developed, the verification of a multi-agent system using MCMAS can have reliability in addition to scalability.

References

References

[1] UK Ministry of Defence. Defence codex. (9), Autumn 2011.

[2] Reg Austin. *Unmanned Aircraft Systems-UAVs Design, Development and Deployment*. John Wiley & Sons, Ltd., 2010.

[3] Federal Aviation Administration. Unmanned aircraft operations in the national airspace system. *Federal Register*, 72(29):6689–6690, February 13, 2007 2007.

[4] R. Alexander, M. Hall-May, and T. P. Kelly. Certification of autonomous systems. Technical Report SEAS/TR/2006/1, SEAS DTC, 2007.

[5] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[6] G. Weiss. *Multiagent Systems: A Modern Approach to distributed Artificial Intelligence*. MIT Press, 1999.

[7] R. W. Beard and T. W. McLain. Multiple uav cooperative search under collision avoidance and limited range. In *Proceedings of the 42nd IEEE Conference on Decision and Control*, pages 25–30, Maui, Hawaii, USA, 2003.

[8] M. Kloetzer and C. Belta. Temporal logic planning and control of robotic swarms by hierarchical abstractions. *IEEE Transactions on Robotics*, 23(3):320–330, April 2007.

[9] S. and E. Frazzoli. Complex mission optimization for multiple-uavs using linear temporal logic. In *2008 American Control Conference*, pages 2003–2009, Westin Seattle Hotel, Seattle, Washington, USA, June 2008.

[10] S. Karaman and E. Frazzoli. Vehicle routing with linear temporal logic specifications: Applications to multi-uav mission planning. In *AIAA Guidance, Navigation and Control Conference and Exhibit*, Honolulu, Hawaii, USA, August 2008.

[11] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Temporal logic motion planning for mobile robots. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pages 2020–2025, 2005.

[12] G. E. Fainekos, H. Kress-Gazit, and G. J. Pappas. Hybrid controllers for path planning: A temporal logic approach. In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, pages 4885–4890, Seville, Spain, December 2005.

[13] M. Webster, M. Fisher, N. Cameron, and M. Jump. Model checking and the certification of autonomous unmanned aircraft systems. In *Proceedings of the 30th International Conference on Computer Safety, Reliability and Security (SAFECOMP)*, September 2011.

[14] J. Chaudemar, E. Bensana, and C. Seguin. Model based safety analysis for an unmanned aerial system. In *DRHE 2010 - Dependable Robots in Human Environments*, Toulouse, France, 16-17 June 2010.

[15] J. R. Abrial. *Modeling in Event-B: System and Software Engineering*. Cambridge University Press, 2010.

[16] T.J. Koo, B. Sinopoli, A. Sangiovanni-Vincentelli, and S. Sastry. A formal approach to reactive system design: Unmanned aerial vehicle flight management system design example. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 522 –527, 1999.

[17] F. Balarin, P. Giusto, A. Jurecska, C. Passerone, E. Sentovich, B. Tabbara, M. Chiodo, H. Hsieh, L. Lavagno, A. Sangiovanni-Vincentelli, and K. Suzuk. *Hardware-Software Co-Design of Embedded Systems: The POLIS Approach*. Kluwer Academic Publishers, 1997.

[18] G. C. Gannod and B. H.C. Cheng. A formal approach for reverse engineering: A case study. In *Proceedings of the 6th working Conference on Reverse Engineering*, pages 100–111, October 1999.

[19] E. M. Clarke, O. Grumberg, and C. Peled. *Model Checking*. MIT Press, Cambridge, Mass, 1999.

[20] G. J. Holzmann. The model checker spin. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

[21] G. J. Holzmann. *The Model Checker SPIN*. Addison-Welsey, 2004.

[22] K. L. McMillan. *Getting started with SMV; User's Manual*. Cadence Berkeley Laboratories, 1998.

[23] G. J. Holzmann. *Design and Validation of Protocols*. Prentice-Hall, 1991.

[24] G. J. Holzmann, P. Godefroid, and D. Pirottin. Coverage preserving reduction strategies for reachability anlysis. In *Proceedings of IFIP/WG 6.1 Symposium of Protocol Specification, Testing, and Verification*, pages 349–364, June 1992.

[25] K. Havelund, M. Lowry, and J. Penix. Formal analysis of a spacecraft controller using spin. *IEEE Transactions on Software Engineering*, 27(8), 2001.

[26] L. R. Humphrey and Z. H. Basnight. Formal modeling, design, and verification techniques for a complex, cooperative uav monitoring task. In *AIAA Modeling and Simulation Technologies Conference*, Portland, Oregon, USA, 8-11 August 2011.

[27] K. L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie-Melon University, 1992.

[28] W. Chan, R. J. Anderson, P. Beame, S. Burns, F. Modugno, D. Notkin, and J. K. Reese. Model checking large software specifications. *IEEE Transactions on Software Engineering*, 24:498–520, July 1998.

[29] J. J. Scillieri, K. R. Butts, and J. S. Freudenberg. Validating executable controller specifications through formal model checking. In *Proceedings of the 2000 IEEE International Symposium on Computer-Aided Control System Design*, pages 179–184, Anchorage, Alaska, USA, September 2000.

[30] Z. Zhao and B. H. Krogh. Formal verification of statechart using finite-state model checkers. *IEEE Transactions on Control Systems Technology*, 14(5):943–950, September 2006.

[31] C. Pecheur and R. Simmons. From livingstone to smv-formal verification for autonomous spacecrafts. In *Proceedings of the First Goddard Workshop on Formal Approaches to Agent-Based Systems (FAABS)*, volume 1871. Springer Verlag, April 2000.

[32] A. Cimatti, C. Pecheur, and R. Cavada. Formal verification of diagnosability via symbolic model checking. In *International Joint Conference on Artificial Intelligence*, volume 18, pages 363–369, 2003.

[33] Y. Choi. From nusmv to spin: Experiences with model checking flight guidance systems. *Formal Methods in System Design*, 30(3):199–216, June 2007.

[34] A. Cimatti, E. M. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sevastiani, and A. Tacchella. Nusmv 2: An opensource tool for symbolic model checking. In *International Conference on Computer-Aided Verificaton*, Copenhagen, Denmark, July 2002.

[35] M. Khalgui and H. Hanisch. Reconfiguration protocol for multi-agent control software architectures. *IEEE Transactions on Systems, Man, and Cybernetics*, 41(1):70–80, January 2011.

[36] Y. Du, C. Jiang, and M. Zhou. A petri net-based model for verification of obligations and accountability in cooperative systems. *IEEE Transactions on Systems, Man, and Cybernetics*, 39(2):299–308, March 2009.

[37] R. H. Bordini, M. Fisher, C. Pardavila, and M. Wooldridge. Model checking agentspeak. In *AAMAS'03*, pages 409–416, 2003.

[38] R. H. Bordini, M. Fisher, W. Visser, and M. Wooldridge. Verifying multi-agent programs by model checking. *Journal of Autonomous Agents and Multi-Agent Systems*, 12:239–256, 2006.

[39] K. Havelund and T. Pressburger. Model checking java programs using java pathfinder. *International Journal on Software Tools for Technology Transfer*, 1999.

[40] R. K. Siminiceanu and G. Ciardo. Formal verification of the nasa runway safety monitor. *International Journal on Software Tools for Technology Transfer (STTT)*, 9(1):63–76, February 2007.

[41] G. Ciardoand, R. L. Jones III, A. S. Miner, and R. K. Siminiceanu. Logic and stochastic modelling with smart. *Performance Evaluation- Modelling techniques and tools for computer performance evaluation*, 63(6):578–608, June 2006.

[42] F. Raimondi. *Model Checking Multi-Agent Systems*. PhD thesis, University of London, 2006.

[43] F. Raimondi, C. Pecheur, and A. Lomuscio. Applications of model checking for multi-agent systems: Verification of diagnosability and recoverability. In *Concurrency Specification and Programming (CSP 2005)*, Ruciane-Nida, Poland, September 2005.

[44] L. Molnar and S. M. Veres. Verification of autonomous underwater vehicles using formal logic. In *Proceedings of the European Control Conference*, pages 1263–1268, 2009.

[45] M. Webster, M.l Fisher, N. Cameron, and M. Jump. Formal methods for the certification of autonomous unmanned aircraft systems. In *Computer Safety, Reliability, and Security: Lecture Notes in Computer Science*, volume 6894/2011, pages 228–242, 2011.

[46] S. Jeyaraman. *Formalising Cooperative Multi-Vehicle Team Behaviours using Kripke Models, Temporal Logic and Model Checking*. PhD thesis, Cranfield University, 2006.

[47] Y. Ke. Dynamic mission planning for multiple unmanned autonomous platforms. Master's thesis, Cranfield University, 2008.

[48] G. Sirigineedi, S. Jeyaraman, A. Tsourdos, R. Zbikowski, and B. A. White. Formal modelling and verification of multi-robot systems specified with temporal logic. *Mathematics in Engineering, Science and Aerospace*, 1(3):255–278, 2010.

[49] C. N. Dean and M. G. Hinchey. Introducing formal methods through role-playing. *ACM SIGCSE (Special Interest Group on Computer Science Education) Bulletin*, 27(1):302–306, March 1995.

[50] M. G.Hinchey and J. P. Bowen. *Applications of Formal Methods*. Prentice Hall, 1995.

[51] J. Rushby. Formal methods and the certification of critical systems. In *Technical report CSL-93-7*. Computer Science Laboratory, SRI International, Menlo Park, USA, December 1993.

[52] N. Storey. *Safety-Critical Computer Systems*. Addison-Wesley Longman Ltd., 1996.

[53] *Information Resources Management-Chapter 1*. US Department of Justice, 2003.

[54] B. S. Blanchard and W. J. Fabrycky. *Systems Engineering and Analysis*. Prentice Hall, 1998.

[55] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlog, 1992.

[56] M. Reniers. *Message Sequence Charts - Syntax and Semantics*. PhD thesis, Eindhoven University of Technology, 1999.

[57] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petit, L. Petrucci, Ph. Schnoebelen, and P. McKenzie. *Systems and Software Verification: Model-Checking Techniques and Tools*. Springer-Verlog, 1999.

[58] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, April 1989.

[59] C. Baier and J. P. Katoen. *Principles of Model Checking*. The MIT Press, 2008.

[60] S. Mauw and M. A. Reniers. High-level message sequence charts. In *Proceedings of the 8th SDL Forum: Time for Testing - SDL, MSC and Trends*, pages 291–306. Elsevier Science Publishers B. V., September 1997.

[61] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[62] J. J. Zhu and R. T. Denton. Timed petri nets and their application to communication protocol specification. In *Proceedings of IEEE Military Communications Conference*, pages 195–199, 1988.

[63] B. Walter. *Timed Petri nets, for Model and Analyzing Protocols with Real-time Characteristics in Protocol Specification, Testing, and Verification III by H. Rudin and C. H. West*. North Holland Publishing Company, 1983.

[64] X. He. Specifying and verifying real-time systems using time petri nets and real-time temporal logic. In *Proceedings on 6th Annual Conference on Computer Assurance - Systems, Integrity, Software Safety and Process Security*, pages 135–140, June 1991.

[65] M. K. Molloy. Performance analysis using stochastic petri nets. *IEEE Transactions on Computers*, 31(9):913–917, September 1982.

[66] G. Logothetis, K. Schneider, and C. metzler. Runtime analysis of synchronous programs for low-level real-time verification. In *Proceedings of the 16th Symposium on Integrated Circuits and Systems Design*, pages 211–216, September 2003.

[67] K. Y. Cai and X. Y. Wang. Towars a control-theoretical approach to software fault-tolerance. In *Proceedings of the 4th International Conference on Quality Software*, pages 198–205, September 2004.

[68] B. F. Chellas. *Modal Logic: An Introduction*. Cambridge University Press, 1980.

[69] P. Blackburn, M. Rijke, and Y. Venema. *Modal Logic - Cambridge Tracts in Theoretical Computer Science 53*. Cambridge University Press, 2001.

[70] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th Annual Symposium on Foundation of Computer Science*, pages 46–57, October 1977.

[71] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons usign branching time temporal logic. In *Proceedings of Logics of Programs Workshop*, May 1981.

[72] E. A. Emerson and J. Y. Haplern. Decision procedures and expressiveness in the temporal logic of branching time. In *Proceedings of 14th ACM Symposium, Theory of Computing (STOC'82)*, pages 169–180, May 1982.

[73] A. Pnueli. The temporal semantics of concurrent programs. *Theoritical Computer Science*, 13(1):45–60, 1981.

[74] J. S. Ostroff. *Temporal Logic for Real-Time Systems, Advanced Software Development Series*. Research Studies Press Limited, 1989.

[75] J. S. Ostroff. Real-time temporal logic decision procedures. In *Proceedings on Real Time Systems Symposium*, pages 92–101, December 1989.

[76] K. R. Apt, N. Francez, and S. Katz. Appraising fairness in languages for distributed programming. *Distributed Computing*, 2:226–241, 1988.

[77] K. G. Larsen, P. Petterson, and W. Yi. Model-checking for real-time systems. In *Proceedings of the 10th International Conference on Fundamentals of Computation Theory*, pages 22–25, August 1995.

[78] X. Nicollin, J. Sifakis, and S. Hovine. Compiling real-time specifications into extended automata. *IEEE TSE Special Issue on Real-Time Systems*, 18(9):794–804, September 1992.

[79] T. A. Henzinger, P. H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[80] D. Beyer, T. A. Henzinger, R. Jhala, and R. Majumdar. The software model checker blast: Applications to software engineering. *International Journal on Software Tools for Technology Transfer*, 9(5):505–525, 2007.

[81] T. Ball and S. K. Rajamani. The slam project: Debugging system software via static analysis. In *Proceedings of the Annual Symposium on Principles of Programming Languages*, pages 1–3, January 2002.

[82] T. Andrews, S. Qadeer, S. K. Rajamani, J. Rehof, and Y. Xie. Zing: Exploiting program structure for model checking concurrent software. In *Proceedings of CONCUR*, pages 1–15, 2004.

[83] F. Lerda and W. Visser. Addressing dynamic issues of program model checking. In *Proceedings of the 8th International SPIN Workshop on Model Checking of Software*, Toronto, Canada, May 2001.

[84] S. Chaki, E. Clarke, A. Groce, S. Jha, and J. Veith. Modular verification of software components in c. *IEEE Transactions on Software Engineering*, 30(6):388–402, June 2004.

[85] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):676–691, 1986.

[86] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[87] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, 27(6):509–516, June 1978.

[88] R. T. Boute. The binary decision machine as a programmable controller. *EUROMICRO Newsletter*, 1(2):16–22, January 1976.

[89] R. E. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM Computing Surveys*, 24(3):293–318, September 1992.

[90] K. S. Brace, R. L Rudell, and R. E. Bryant. Efficient implementation of a bdd package. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 20–25. IEEE Computer Society Press, 1990.

[91] UK Ministry of Defence. Defence codex-special supplement. (2), Autumn 2008.

[92] G. Watson, A. Whitbrook, M. Kaye, and K. Hobbs. Systems exemplar 2 - land transit. In *SEAS DTC Proceedings of Fifth Annual Conference*, Edinburgh, UK, July 2010.

[93] S. Kim, J. Choi, and Y. Kim. Fault detection and diagnosis of aircraft actuators using fuzzy-tuning imm filter. *IEEE Transactions on Aerospace and Electronic Systems*, 44(3):940–952, July 2008.

[94] N. Cameron, M. Webster, M. Jump, and M. Fisher. Certification fo a civil uas: A virtual engineering approach. In *AIAA Modelling and Simulation Technologies Conference*, Portland, Oregon, USA, 8-11 August 2011.

[95] S. Kim, P. Silson, A. Tsourdos, and M. Shanmugavel. Dubins path planning of multiple uavs for communication relay. *Journal of Aerospace Engineering, Proceedings of the Institution Mechanical Engineers Part G*, 225(1):12–25, January 2011.

[96] G. Sirigineedi. *Formally Verified Distributed Decision Making Strategies for Multiple Autonomous UAVs using Kripke Models and Model Checking*. PhD thesis, Cranfield University, 2011.